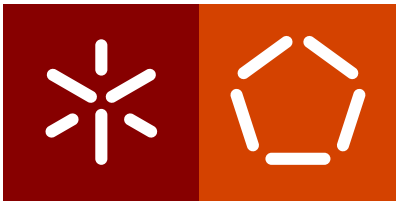


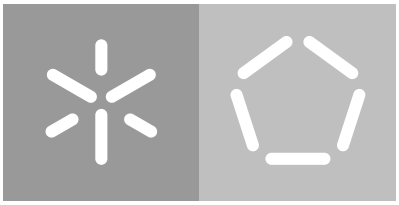
Universidade do Minho
Escola de Engenharia
Departamento de Eletrónica Industrial

Sandra Manuela Gonçalves Dixe

Deteção de Danos no Interior de um Carro por Visão por Computador

Damage detection inside a car using Computer Vision

Guimarães, Novembro de 2020



Universidade do Minho
Escola de Engenharia
Departamento de Eletrónica Industrial

Sandra Manuela Gonçalves Dixe

Deteção de Danos no Interior de um Carro por Visão por Computador

Damage detection inside a car using Computer Vision

Dissertação de Mestrado
Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor José Araújo Mendes
Doutor João Borges

Guimarães, Novembro de 2020

RESUMO

Atualmente, os veículos totalmente autónomos são vistos como o futuro da indústria automóvel, perspectivando-se assim uma revolução na forma como os meios de transporte são hoje em dia utilizados. Com o desenvolvimento deste meio de transporte, o conceito de *Shared Autonomous Vehicles (SAV)* ganha uma grande relevância uma vez que possibilita o transporte de uma forma fácil, rápida e económica. Para garantir que os SAV fornecem todas as condições de conforto e limpeza para serem usados pelas pessoas, é necessário a utilização de um sistema de monitorização para averiguar o estado do interior do carro, nomeadamente para verificar a presença de sujidade e danos (i.e. corte, desgaste, queimaduras e manchas). Foi objetivo deste trabalho avaliar possíveis soluções para solucionar a deteção de sujidade e danos dentro de um carro. Neste trabalho foram desenvolvidos algoritmos de *Deep Learning* para a classificação de sujidade e danos no interior de veículos automóveis. Para além da classificação destes danos é também de extrema importância saber a sua distribuição espacial.

O trabalho é constituído por três módulos: Seleção e Estudo de Sensores para Caracterização de Danos, Desenvolvimento de Algoritmos para Deteção de Danos e Sujidade e por fim, Avaliação dos Algoritmos no Interior de Veículos Automóveis.

No módulo Seleção e Estudo dos Sensores para Caracterização dos Danos, é apresentado o estudo dos tipos de sensores de imagem que melhor se adequam ao problema.

No módulo do Desenvolvimento de Algoritmos para Deteção de Danos e Sujidade é apresentado o estudo de artigos e métodos para deteção de danos. Depois do estudo e comparação entre várias abordagens do Estado de arte foram escolhidos métodos de *Deep Learning*: Deteção de Objetos e *Semantic Segmentation*, para a resolução da nossa problemática. Ainda neste módulo, os algoritmos desenvolvidos são avaliados recorrendo a um *dataset out-car* anteriormente criado.

Por último, é apresentado o módulo Avaliação dos Algoritmos no Interior de Veículos Automóveis, no qual com base no comportamento obtido dos algoritmos avaliados no *dataset out-car*, são escolhidos e avaliados os melhores algoritmos num *dataset in-car* também criado, os métodos que obteram as melhores performances foram os métodos de *Semantic Segmentation*: DeepLabV3+ e U-Net. O modelo DeepLabV3+ mostrou resultados muito promissores na deteção e classificação desta problemática, obtendo resultados muito satisfatórios para as classes bom, dano, mancha e sujidade, com uma Exatidão de 77.17%, 58.60% 65.81% e 68.82%, respetivamente.

Palavras chave: Shared Autonomous Vehicles, Deep Learning, Supervised Learning, Semantic Segmentation, Object Detection.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração. Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

ABSTRACT

Currently, fully autonomous vehicles are seen as the future of the automotive industry, thus envisaging a revolution in the way in which transport is used today.

With the development of this means of transport, the concept of Shared Autonomous Vehicles SAV gains great relevance since it allows transportation in an easy, fast and economical way. To ensure that the SAVs provide all the conditions of comfort and cleanliness to be used by people, it is necessary to use a monitoring system to check the status of the interior of the car, namely to check for the presence of dirt and damage (ie cut, wear, burns and stains). The objective of this work was to evaluate possible solutions to solve the detection of dirt and damage inside a car. In this work, Deep Learning algorithms were developed for the classification of dirt and damage inside vehicles. In addition to the classification of these damages, it is also extremely important to know their spatial distribution.

The work consists of three modules: Selection and Study of Sensors for Characterization of Damage, Development of Algorithms for the Detection of Damage and Dirt and finally, Evaluation of Algorithms in the Interior of Vehicles.

In the Selection and Study of Sensors for Damage Characterization module, the study of the types of image sensors that best suit the problem is presented.

In the Development of Algorithms for the Detection of Damage and Dirt module, is presented the study of articles and methods for detecting damage. After studying and comparing several state-of-the-art approaches, Deep Learning: Object Detection and Semantic Segmentation methods were chosen to solve our problem. Also in this module, the developed algorithms are evaluated using an out-car dataset previously created.

Finally, the module Evaluation of Algorithms in the Interior of Vehicle is presented, in which based on the behavior obtained from the algorithms evaluated in the out-car dataset, the best algorithms are chosen and evaluated in an in-car dataset also created, the methods that obtained the best performances were the Semantic Segmentation methods: DeepLabV3 + and U-Net. The DeepLabV3+ model showed very promising results in the detection and classification of this problem, obtaining very satisfactory results for the classes good, damage, stain and dirty, with an Accuracy of 77.17%, 58.60% 65.81% and 68.82%, respectively.

Keywords: Shared Autonomous Vehicles, Deep Learning, Supervised Learning, Semantic Segmentation, Object Detection.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

ACRÓNIMOS

A

ASPP Atrous Spatial Pyramid Polling.

C

CNN Convolutional Neural Networks.

D

DL Deep Learning.

E

ELM Extreme Learning Machine.

F

FAST R-CNN Fast Region - based Convolution Neural Networks.

FASTER R-CNN Faster Region - based Convolution Neural.

FCL Fully Connected Layer.

FCN Fully Convolution Network.

J

JSON JavaScript Object Notation.

M

ML Machine Learning.

O

OD Object Detection.

R

R-CNN Region - based Convolution Neural.

RCT Redundant Contourlet Transform.

ROI Region of Interest.

S

SAV Shared Autonomous Vehicles.

SS Selective Search.

SSD Single Shot Detector.

SVM Support Vector Machine.

T

TILDA Textile Texture-Database.

TND Teste Não Destrutivo.

U

URE Unidades Repetitivas Elemantares.

Y

YOLO You Only Look Once.

1

INTRODUÇÃO

Neste capítulo inicial dá-se a conhecer o âmbito do projeto, o seu enquadramento na sociedade e o estado de trabalhos desenvolvidos em áreas semelhantes. Refere-se do mesmo modo a importância dos progressos realizados na área, as partes interessadas e qual o objetivo final da dissertação.

1.1 Enquadramento

Com o rápido desenvolvimento da tecnologia no sector da condução autónoma, a forma como atualmente conhecemos e interpretamos os meios de transporte pode mudar radicalmente. Segundo Fragnant and Kockelman Fragnant and Kockelman [2018], esta tecnologia deve estar comercialmente desenvolvida até ao fim da década. As vantagens que este transporte pode trazer parecem ser muitas, tal como, uma maior segurança e um menor impacto ambiental, Littman and Paluck [2015]. Em contra partida, não se pode desvalorizar o caminho que ainda falta até este meio de transporte estar efectivamente disponível.
Tettamanti et al. Tettamanti et al. [2016], argumenta que o período de transição, no qual coexistem veículos autónomos e veículos convencionais, é bastante complexo, porque a gestão de comportamentos de navegação precisos, através de condução autónoma, contra comportamentos irracionais apresenta um nível de complexidade elevado. Krueger et al. Krueger et al. [2016] refere que tanto estudantes académicos como profissionais declaram que a introdução de SAV é uma boa abordagem para que esta tecnologia ganhe popularidade, pois assim, é possível fornecer a um alargado número de pessoas mobilidade deste nível a baixo custo. Estes serviços já estão a ser pensados há muito tempo, notando-se uma evolução da própria designação, começando por, *robotaxis*, *self-driving taxis*, Fagnant and Kockelman [2014], *aTaxis*, por último o termo SAV é a designação usada hoje em dia e que está em grande emersão, dada por Fagnant and Kockelman [2015].
Os serviços SAV, permitem que o consumidor aceda a um conjunto de veículos autónomos pertencentes a um proprietário em troca de uma taxa de uso, Cohen and Kietzmann [2014]. Por exemplo, a Volkswagen e a Ford pretendem lançar esse serviço até 2021, Fagnant and Kockelman [2014]. A ideia é aceder a estes serviços através de aplicações de telemóvel, em vez de ter que procurar ou caminhar longas distâncias até um veículo disponível. Presume-se que estes SAV sejam totalmente autónomos, sem necessidade de intervenção humana.
Este novo conceito de mobilidade origina um novo paradigma no fator humano. Com a ausência de condutor no veículo automóvel deixa de existir um responsável pelo espaço. Dessa forma surge a necessidade de desenvolver sistemas avançados de monitorização do interior do veículo, podendo assim garantir o máximo conforto e integridade aos passageiros. Relativamente ao conforto do veículo, podem ser desenvolvidos sis-

temas de deteção de danos e sujidade, garantindo assim a segurança do veículo bem como a qualidade do serviço prestado.

1.2 Motivação e importância

Os trabalhos desenvolvidos nesta área são movidos por vários objetivos, tais como: deteção de danos em variados ambientes e nos mais diversos materiais, Liu et al. [2010]; Montanini [2010]; Furtado et al. [2001], através das mais diversas abordagens, Jing et al. [2013]; Hu [2014]; Malek et al. [2013]; H İbrahim Çelik [2014]. Os estudos que se tornaram relevantes para o desenvolvimento deste documento incidiram sobre a deteção de danos em tecido e a classificação dos mesmos. Alguns destes estudos apresentam ainda, localização espacial destes danos, Yapi et al. [2018]; Liu et al. [2019]; Jeyaraj and Samuel Nadar [2019]. As abordagens apresentadas nestes estudos de inspeção de qualidade industrial são de interesse para o estudo apresentado.

No que toca às possíveis partes interessadas destaca-se a indústria na generalidade, com foco na indústria têxtil ou do calçado. A existência de um dano visível no produto, torna-o inútil para venda. Ou então, noutros casos, faz com que o preço caia abruptamente de modo que, pode até nem cobrir o valor investido na matéria prima. Tudo isto, é um motivo de preocupação para as industrias, daí a necessidade de estudar e investigar continuamente novas metodologias de correção e prevenção de danos, o mais preventivamente possível.

Existem artigos e estudos desenvolvidos até ao momento direccionados na classificação de danos em muitos sectores, mas a vertente de exploração de danos em tecido no ambiente de interior de um carro não foi ainda explorada. O facto de um carro possuir características únicas de luminosidade e proporcionar áreas condicionadas que se podem tornar de difícil inspecção torna este estudo muito interessante. Neste trabalho os veículos que tem interesse analisar são os SAV.

Com o passar do tempo e com o rápido desenvolvimento da engenharia o conceito dos SAV vai passar rapidamente a ser uma realidade do nosso dia a dia. Dito isto, torna-se deveras interessante e motivacional fazer um estudo e desenvolver algoritmos capazes de detetar danos e sujidade no interior de um veículo automóvel. Um SAV tornar-se-á um local público (compartilhado pelos passageiros) em que é necessário garantir um espaço acolhedor, cómodo, confortável para que seja a primeira opção quando se pensa em fazer uma viagem, seja esta, de grande curso ou não. Sítios que transmitam confiança e acima de tudo máxima segurança são valorizados por todos. É de máximo interesse para o dono do SAV que este seja apresentado sempre de forma irrepreensível aos passageiros.

Em suma, a motivação deste trabalho foca-se no estudo e desenvolvimento de um algoritmo que seja capaz de detetar e classificar danos no interior de um SAV.

1.3 Objetivo

A área de trabalho desta dissertação diz respeito à análise do interior do carro. Os veículos ao fim de um número de viagens podem sofrer um certo desgaste. Também os passageiros, de forma intencional ou involuntária, podem danificar o interior dos carros, por exemplo pontas de cigarros que podem provocar pequenas queimaduras, ou então algum elemento cortante, por exemplo através de um acessório de uma

senhora, pode provocar perfurações no interior do carro. Estes são pequenos exemplos de um variado leque de situações que podem acontecer no decorrer de várias viagens sequenciais e com variadíssimas pessoas juntas num mesmo espaço (interior do carro). Considerando os objetivos apresentados, este trabalho foi dividido em três módulos sequenciais, ilustrados na Figura 1.

O primeiro módulo diz respeito ao estudo e seleção dos principais sensores utilizados para a deteção das anomalias no interior do carro. O sensor é tanto melhor quanto maior o contraste que se obtiver entre dano e parte saudável do tecido. Ainda neste módulo, os sensores seleccionados serão utilizados para a gravação de imagens de amostras de tecido com danos e imagens do interior de vários carros (com e sem danos) com o objetivo de criar dois *datasets*: *dataset out-car* e *dataset in-car*. Quanto maiores e mais diversificados forem estes *datasets* mais precisos se tornam os algoritmos de *Deep Learning* neles testados posteriormente.

O segundo módulo é a seleção e customização de algoritmos para deteção de danos no interior de um SAV, neste módulo serão apresentados potenciais algoritmos para a deteção de danos e sujidade em tecido recolhidos do estado da arte. Posteriormente será feita a seleção e implementação dos que apresentem maior probabilidade de sucesso no contexto da problemática apresentada. Por fim, estes serão avaliados com o *dataset out-car* criado no primeiro módulo.

Por último, no terceiro módulo serão avaliados os melhores algoritmos do módulo anterior, no *dataset in-car*. Esta avaliação será suportada de ensaios adicionais, de ablação e hiperparâmetros, no sentido de seleccionar o algoritmo final a utilizar.

Figura 1: Representação em diagrama de blocos do sistema geral para uma obtenção um algoritmo eficaz na deteção e classificação de danos no interior de um carro.

1.4 Estrutura da dissertação

Esta dissertação encontra-se divida em seis capítulos, estes capítulos encontram-se organizados e sequenciados de forma a que o leitor consiga compreender de forma simples todo o trabalho desenvolvido, acompanhando assim detalhadamente todo o processo realizado.

O primeiro capítulo é constituído pela Introdução, que serve para fazer o enquadramento do projeto desenvolvido, expondo os objetivos, o propósito da dissertação e a sua estrutura.

O segundo capítulo, Estado da arte, irá explorar e relatar todos os conceitos envolvidos na criação deste projeto. Neste capítulo serão apresentados documentos bibliográficos de interesse directo para a resolução deste projeto. São apresentados ainda, os materiais mais utilizados no fabrico de interiores de automóveis

e os danos tipicamente encontrados dentro destes. Seguidamente são apresentadas algumas abordagens multi-espectrais (i.e. IR, RGB), possibilitando assim a seleção dos sensores a utilizar no capítulo 3. Por último são apresentados métodos que se mostraram eficientes na deteção destes danos em diferentes ambientes, podendo estes métodos tornarem-se bons candidatos na resolução desta problemática.

O terceiro capítulo, Estudo e seleção dos sensores para caracterização dos danos out-car, contém a seleção dos sensores a utilizar na captura de *features*, bem como a explicação detalhada de todo o processo efetuado na geração dos *datasets out-car* e *in-car*.

O quarto capítulo, Desenvolvimento de algoritmos para deteção de danos, é dedicado à explicação dos algoritmos desenvolvidos para a deteção de danos, estes algoritmos são testados inicialmente no *dataset out-car*. Por último, é feita uma discussão e consequente validação dos algoritmos que apresentarem as melhores performances.

O quinto capítulo, Avaliação dos algoritmos no interior do veículo, é dedicado à avaliação no *dataset in-car* dos melhores algoritmos resultantes do capítulo quatro. Neste capítulo, é possível efectuar algumas apreciações gerais do trabalho desenvolvido, e serão discutidas e justificadas algumas das escolhas efectuadas durante esta avaliação de algoritmos no contexto da problemática *in-car*. Por último, é feita uma discussão e consequente validação dos algoritmos que apresentarem as melhores performances.

O sexto capítulo, Conclusão, é apresentada a conclusão final da dissertação e são apresentadas as ideias para trabalhos futuros.

Na parte final da dissertação são apresentados os Anexos, que contém o pseudocódigo de alguns dos algoritmos desenvolvidos.

2

ESTADO DE ARTE

Neste capítulo, é apresentada a análise de documentos bibliográficos de interesse para o desenvolvimento desta dissertação.

Com o rápido desenvolvimento da engenharia, a indústria está a avançar para a era da Indústria 4.0. As produções inteligentes tornaram-se uma tendência inevitável no seu desenvolvimento. Geralmente, na produção industrial, seja ela de que natureza for, o controlo da qualidade dos produtos produzidos é uma questão muito pertinente.

Normalmente, a inspecção humana é usada para a detecção de defeitos, esta consegue fornecer uma correção instantânea de pequenos defeitos, mas em contrapartida a inspecção humana pode não detectar erros devido ao descuido ou ilusão ótica. A inspecção humana falha muitas vezes na deteção em termos de precisão, consistência e eficiência, pois os trabalhadores estão sujeitos ao tédio, originando assim resultados de inspecção com elevada incerteza.

Assim sendo, a inspecção automatizada torna-se um método eficiente para melhorar a qualidade do produto. Dentro deste tema foram e são atualmente desenvolvidos vários estudos no sentido de investigar um ou mais métodos/metodologias que sejam realmente eficazes neste controlo. Nas seguintes secções estão apresentados muitos dos estudos publicados até agora para resolução desta problemática.

2.1 Materiais mais encontrados no interior dos automóveis

Os tecidos/materiais usados nos carros são uma parte importante no que se refere à estética e conforto do seu interior, além disso, forros e revestimentos em bom estado podem ser um diferencial no momento da revenda. As tecnologias atualmente desenvolvidas na produção de tecidos permitem que exista uma vasta gama de opções disponíveis no mercado, ver Figura 2.

As opções existentes de materiais para uso nos interior de veículos, GrupoJBtecidos [2016], são:

- **Couro**
  - Material natural nobre, durante muito tempo foi associado aos modelos de carros de segmento de alto padrão. Embora hoje em dia possa ser encontrado em modelos mais populares, ainda remete a elegância e sofisticação.
  - Resistente e confortável. Conquista adeptos em todo o mundo, é dos revestimentos mais procurados para a reforma no interior dos automóveis.
  - É um tecido de alto custo e que exige cuidados na manutenção, como a hidratação periódica, para conservar as suas características e evitar assim possíveis rachaduras.

- **Courvin**
  - O courvin pode ser considerado a versão sintética do couro, outro nome dado é o vinil.
  - Com a evolução das técnicas de produção, os resultados deste tecido podem mesmo chegar a superar o couro natural, conquistando assim cada vez mais espaço na indústria automóvel.
  - É considerado bastante resistente, confortável e com apelo estético, tornando-se assim uma ótima opção para aqueles que buscam uma alternativa de menor custo ao couro natural.

- **Tecido navalhado**
  - São tecidos aveludados acoplados a uma espuma de diferentes gramaturas, o que garante um grande conforto no interior dos veículos, ao mesmo tempo que oferece boa resistência ao atrito.
  - O tecido navalhado permite reproduzir quase todos os padrões existente. Torna-se por isto, uma excelente alternativa para reformar o interior do seu automóvel sem abrir mão das características originais.

- **Tecido feito em Tear**
  - Utilizado atualmente pela indústria de automóvel, apresenta um ótimo custo benefício, suportando as mais diferentes estampas, está presente em veículos de diversas categoriais.
  - Pode ser encontrado com acoplagem de espuma que aumenta o conforto e resistência em especial nos encostos de banco. Além disso, as suas características não são alteradas com as variações de temperatura e é de fácil higienização.

- **Malharia**
  - Material mais utilizado atualmente, possuí um fio muito fino na sua confecção, garantindo assim um aspecto mais suave ao toque e esteticamente menos rústico em comparação a outros materiais.
  - Também pode ser encontrado em diferentes cores e estampas, sendo ideal para aplicação em diferentes modelos de automóveis, garantindo conforto e boa resistência.

(a) (b) (c) (d) (e)

Figura 2: Materiais utilizados no interior de veículos. (a) Couro; (b) Courvin; (c) Tecido Navalhado; (d) Tear; (e) Malharia.

2.2 Danos tipicamente encontrados dentro dos veículos automóveis

Lavrador Lavrador [2019] conclui que, a deteorização que ocorre precocemente dentro dos veículos está relacionada com certos comportamentos que os seus ocupantes apresentam sistematicamente no dia a dia. O interior dos veículos é maioritariamente constituído por plástico e tecido, sendo que os tecidos podem ser de algodão ou sintéticos, imitando pele. O autor refere como principal responsável para a degradação dos plásticos e dos tecidos tipo "pele", os desinfectantes que as pessoas usam para matar as bactérias das mãos, porque estes produtos contêm muito etanol que deteora agressivamente estes materiais. Exemplos de danos no interior dos veículos podem ser vistos na Figura 3.

Outro produto extremamente prejudicial para os materiais que revestem o habitáculo dos carros é o protector solar com elevado fator de proteção. Os químicos constituintes do protector solar protegem a pele humana dos raios solares, mas estão longe de fazer o mesmo em relação aos plásticos. Por fim, outro químico que não é particularmente simpático com os revestimentos do habitáculo do veículo são os repelentes muitas vezes usados no verão para maior comodidade, repelindo assim certos insetos.

Tortas Tortas [2019], refere que o tablier do carro é geralmente uma zona onde predominam os plásticos, e o facto de esta zona estar bastante exposta ao sol faz com que estes plásticos acabem por perder rigidez e até partirem-se. Nos carros com materiais de menor qualidade ao fim de apenas alguns anos começam a notar-se alguns sinais de desgaste. O mesmo se aplica aos estofos, que facilmente podem apresentar manchas ou sinais de desgaste. Já nos tapetes o que aparece mais frequentemente é sujidade ou sinais de ferrugem devido à humidade do calçado no inverno.

A Ford Europa Gonçalves [2019], decidiu pôr em prática uma série de testes, que têm como objetivo, testar como é que os bancos dos seus carros, resistem ao suor humano. O suor humano tem propriedades, que quando entra em contacto com os diferentes materiais que compõem os bancos dos carros, origina desgaste precoce.

(a) (b) (c)

Figura 3: Representação de danos que podem ocorrer no habitáculos dos veículos. (a) Corte no tecido. (b) Vários cortes em pele sintética devido ao desgaste e ressecamento. (c) Interior da porta com visível perda de cor devido à excessiva exposição solar.

2.3 Testes não destrutivos

Em todos os trabalhos estudados, o foco é procurar e identificar qual o objetivo do documento, e entender o modelo ou método utilizado. Por outro lado, é importante perceber quais os sensores usados para a reso-

lução do problema deteção e inspecção de defeitos nas diversas áreas da indústria com foco na inspecção têxtil. É também importante perceber a interpretação dos métodos de inspecção e por fim analisar o *setup* experimental.
Ao longo desta análise retiram-se conhecimentos/conclusões importantes que podem ser posteriormente aplicados no desenvolvimento desta dissertação. Dentro deste subcapítulo são apresentadas várias abordagens de inspecção com foco no uso de sensores RGB, existindo também a hipótese de inspecção recorrendo à termografia. Tanto a inspecção termográfica como a inspecção recorrendo a câmaras RGB são considerados um *Teste Não Destrutivo (TND)*.
Denomina-se de TND a qualquer tipo de teste efetuado a um material que não altere nem interfira de nenhuma maneira nas suas propriedades físicas, químicas, mecânicas ou dimensionais. Os testes não destrutivos implicam um dano impercetível ou nulo preferencialmente.
Os TND traduzem-se num vasto leque de técnicas de análise utilizadas na ciência e na indústria para avaliar as propriedades de um material, componente ou sistema, sem causar danos, baseando-se na aplicação de fenómenos físicos tais como ondas eletromagnéticas, acústicas, elasticidade, emissão de partículas subatómicas, capilaridade, absorção e qualquer tipo de teste que não implique um dano considerável à amostra examinada.

2.3.1 *Inspecção termográfica*

O uso da termografia é um TND, baseada no princípio de que dois materiais diferentes, ou seja, possuindo propriedades termofísicas diferentes, produziriam duas assinaturas térmicas distintas que podem ser reveladas por um sensor infravermelho, como uma câmara térmica, Ibarra-Castanedo et al. [2013]. A termografia é uma técnica frequentemente usada na deteção de danos na indústria.
Uma sub-abordagem termográfica de TND que permite deteção de defeitos superficiais e internos é a termografia *lock-in*. Através desta abordagem é possível detetar defeitos superficiais em materiais acrílicos, Liu et al. [2010]; Montanini [2010].

2.3.2 *Inspecção RGB*

No documento, referenciado por Furtado et al. Furtado et al. [2001] foram implementadas técnicas de processamento de imagem para a detecção de defeitos em tecidos. A partir de uma imagem em níveis de cinzento foi possível obter uma imagem binária que descrimina o defeito, após a aplicação de uma segmentação por *threshold*, análise estatística e morfológica matemática. Foram realizados testes e obtidos resultados satisfatórios para os seguintes tipos de defeitos em tecido: fio grosso, falta de fio, borboto, mancha e nódoa. Neste trabalho foram estudadas técnicas para a detecção de defeitos estáticos, usando algoritmos simples e com um reduzido tempo de processamento. A partir de imagens em níveis de cinzento foi aplicado o operador morfológico erosão, obtendo-se uma imagem binária que contém o defeito. As experiências descritas foram testadas no programa MATLAB, posteriormente foram implementadas num sistema de detecção de defeitos em tecidos, desenvolvido usando o MicrosoftTM Visual C++, baseado em programação por objectos.
Como *setup* experimental foi utilizado um sistema de aquisição de imagem composto por uma câmara, um

adaptador e um computador. Durante a realização deste trabalho, foi necessária a construção de uma câmara escura, para que não houvesse interferências com a luz exterior. Esta foi composta, por um sistema de iluminação, o qual é constituído por uma lâmpada fluorescente (Philips F18W/54) e um balastro electrónico. A lâmpada encontrava-se a uma altura de 80 cm com um ângulo de incidência de 16°.

Uma das metodologias mais utilizada para deteção e classificação de defeitos é baseada em filtros de Gabor Jing et al. [2013]; Hu [2014].

Jing et al. Jing et al. [2013] propôs um método de detecção e classificação de defeitos com base em filtros de Gabor e no modo binário local. No entanto, este modo tem algumas falhas. O modo binário local depende fortemente do estabelecimento do valor do *threshold*, se não for definido corretamente, algumas regiões normais próprias da textura da fibra têxtil são erradamente detectadas como regiões defeituosas.

Hu Hu [2014] propõe um método de filtro de Gabor baseado na otimização do algoritmo de *simulated annealing*, que transforma o filtro em uma frequência e direção específica para corresponder ao recurso de imagens não defeituosas. No entanto, nesses métodos baseados em filtros Gabor, o cálculo de seus parâmetros precisa ser muito preciso, detalhado e computacionalmente intensivo, dificultando a obtenção de resultados de avaliação realmente bons.

Contudo outras abordagens tem vindo a ser testadas. Raheja et al. Raheja et al. [2013] propôs um método para detectar e avaliar defeitos de fibras têxteis com base na matriz de Co-ocorrência de nível de cinza. Este método tem alto custo computacional. Em suma, este método apresenta bons resultados para detecções em pequena escala.

Malek et al. Malek et al. [2013] propõe um método para detectar e avaliar automaticamente defeitos têxteis usando métodos rápidos de correlação cruzada combinada de *Fourier*. No entanto, estes métodos não conseguiram detectar efetivamente defeitos na imagem no domínio espacial e nem detetar defeitos em imagens com texturas aleatórias.

H Ibrahim Celik H İbrahim Çelik [2014] combinou filtragem linear com operações morfológicas para propor um método de detecção e avaliação de defeitos em fibras têxteis. Contudo, a configuração de *threshold* do método é muito rigorosa e o desempenho no geral é baixo.

O documento referenciado por Yapi et al. Yapi et al. [2018], apresenta uma abordagem baseada em aprendizagem para a detecção automática de defeitos em tecido. A abordagem proposta é baseada numa representação estatística de padrões de tecido usando a *Redundant Contourlet Transform (RCT)*. A distribuição dos coeficientes de RCT é modelada usando uma mistura finita de gaussianos generalizados, que constituem assinaturas estatísticas que fazem a distinção entre tecidos defeituosos e sem defeitos. Além de compactas e rápidas de calcular, essas assinaturas permitem ainda a localização precisa dos defeitos.

Numa primeira fase, é aplicado um pré-processamento que detecta o tamanho básico do padrão para assim decompor a imagem e calcular a assinatura. Na segunda fase, são usadas amostras de tecido rotuladas para treinar um classificador Bayes que consegue discriminar entre tecidos com e sem defeito.

A abordagem proposta promete lidar com vários tipos de tecidos, dos mais simples aos mais complexos. Experiências recorrendo ao *dataset* TILDA, demonstraram que este método produz melhores resultados comparando com métodos mais recentes.

Neste artigo, é então apresentado um novo algoritmo de detecção de defeitos que tem a capacidade de lidar com diferentes tipos de defeitos, como exemplo, são apresentadas dois conjuntos de amostras denominados de p1 e não p1, ver Figura 4 e Figura 5, respetivamente.

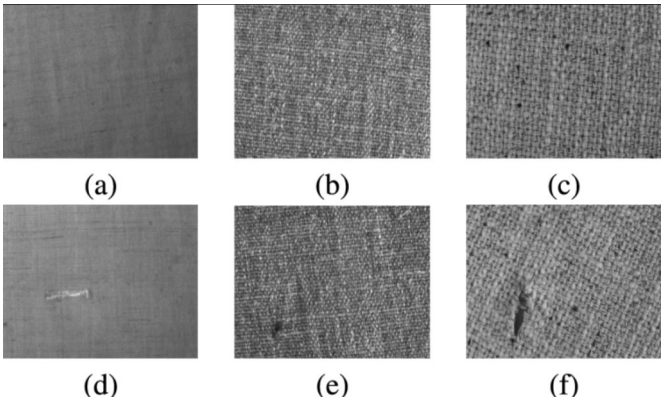


Figura 4: Exemplo do banco de dados TILDA que representa o grupo p1. (a) Tecido liso sem defeitos. (b) Tecido de sarja sem defeitos. (c) Tecido liso sem defeitos. (d) Tecido liso com defeitos. (e) Tecido de sarja com defeitos. (f) Tecido liso com defeitos. Imagem adaptada de Yapi et al. [2018].

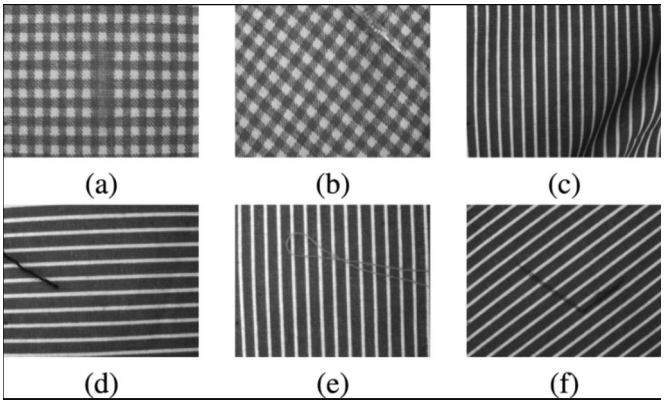


Figura 5: Exemplo do banco de dados TILDA que representa o grupo Não-p1. a) Tecido com padrão. (b) Tecido oblíquo com padrão. c) Tecido com faixas verticais. (d) Tecido com faixas horizontais. (e) Tecido com listras verticais. (f) Tecido com listras oblíquas. Imagem adaptada de Yapi et al. [2018].

A técnica usada consiste na decomposição da imagem em *Unidades Repetitivas Elemantares (URE)*, caracterizadas por uma periodicidade ao longo de uma amostra de tecido, ver exemplo na Figura 6.

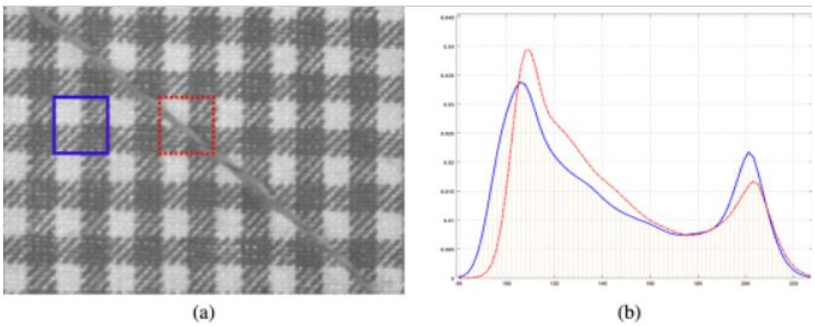


Figura 6: Cálculo da assinatura RCT-MoGG para tecidos com defeito e sem defeitos. (a) Exemplos de blocos defeituosos (quadrado vermelho tracejado) e livres de defeitos (quadrado azul sólido). (b) As linhas tracejadas de vermelho e azul sólido mostram assinaturas MoGG dos blocos defeituosos e sem defeitos, respectivamente. Imagem adaptada de Yapi et al. [2018].

O método de detecção de defeitos é baseado em *learning-based approach* e separa assinaturas RCT-MoGG

de tecidos sem defeitos e tecidos defeituosos. Após a decomposição de uma imagem de tecido em URE, uma classificação dos blocos de URE é realizada para avaliar se cada URE está livre de defeitos ou não. Este método de análise fornece uma alta precisão geral de detecção e uma taxa de falsos positivos muito baixa, em comparação com métodos recentemente propostos.

Metodologias mais recentes apoiam-se em técnicas de ML, Liu et al. [2019] e *Deep Learning (DL)*, Jeyaraj and Samuel Nadar [2019].

Liu et al. Liu et al. [2019], é introduzido um novo método para classificação de imagens de tecido com defeitos recorrendo à segmentação não supervisionada, o novo método é o *Extreme Learning Machine (ELM)* e promete equilibrar a eficiência e a precisão no reconhecimento de defeitos.

O artigo reconhece que nas últimas três décadas foram apresentados inúmeros métodos de detecção de defeitos em tecido, recorrendo a técnicas de visão computacional e reconhecimento de padrões. Os métodos mais conhecidos e utilizados são a análise relacional de *Gray*, os coeficientes de transformada *Wavelet*, transformação de Fourier, filtros Gabor e transformação redundante de contorno.

Estes métodos reconhecem defeitos por meio de uma extração de características da textura do tecido, a sensibilidade da detecção pode ser afectada quando os defeitos são muito pequenos e com baixo contraste. Segundo o artigo, os principais desafios são a detecção de defeitos em certas malhas, pois estas incluem a complexidade das texturas.

O *pipeline* do método proposto neste documento está representado na Figura 7. Ao contrário do uso de modelos adaptativos, este método foca na segmentação não supervisionada, que é útil para facilitar a extração de recursos geométricos. A abordagem proposta inclui quatro partes: segmentação de defeitos, extração de features, treino do classificador ELM e fusão de probabilidade Bayesiana.

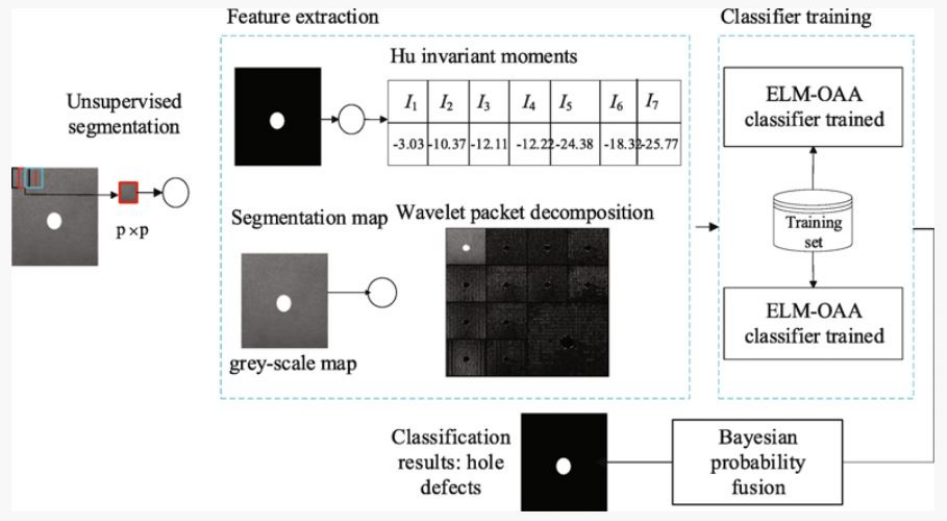


Figura 7: O pipeline inclui segmentação de defeitos, extração de recursos, treino do classificador ELM e fusão de probabilidade bayesiana. A ideia principal deste trabalho é apresentar uma segmentação não supervisionada e um método ELM para classificação de defeitos de tecido. Imagem adaptada de Liu et al. [2019].

Todos os testes foram realizados num computador com uma CPU Intel Core i5 de 3.3GHz e 8GB de RAM DDR3. A maior parte do código foi implementada na plataforma de software MATLAB.

O autor considera a deteção e classificação de defeitos de superfície em tecidos comuns um problema desafiador. Todo o processo de classificação de defeitos de tecido é trabalhoso e computacional. Para contrariar

isso, neste artigo, é apresentado um novo método de classificação de defeitos de tecido. Este modelo foi avaliado recorrendo ao *dataset* TILDA e em algumas amostras de tecido reais.
Os resultados demonstram a eficácia do método na detecção de defeitos de várias formas, tamanhos e locais. A precisão da classificação do método apresentado é de 91.8%, o que o torna melhor que os outros métodos. Na Figura 8, é apresentada uma tabela comparativa entre o método usado e outros métodos estudados no artigo.

Performance	Machine	Cao's [5]	Jing's [30]	Zhang's [53]	Our method
Data source	Production	TILDA	TILDA	Textile factory	Textile factory
Theory	Real-design	PG-LSR	Gabor filter	RBF network	ELM
Complexity	High	Medium	High	High	Low
Cost	High	Medium	High	High	Low
Efficiency	High	Medium	Low	Medium	High
Accuracy	High	Medium	Medium	Medium	Medium

Figura 8: Comparação entre métodos. Imagem adaptada de Liu et al. [2019].

Jeyaraj and Samuel Nadar Jeyaraj and Samuel Nadar [2019], propõe um modelo que permite detetar com precisão a região defeituosa e a natureza dos possíveis defeitos presentes em tecidos. O artigo propõe uma abordagem de DL usando uma *Convolutional Neural Networks (CNN)*, sendo que este algoritmo classifica os defeitos de forma não supervisionada. O artigo refere que existem dois grandes grupos de defeitos que podem ser encontrados em tecido: defeitos de cor e alteração da textura do tecido. As experiências realizadas focaram-se nesses dois grupos de defeitos. A contribuição geral do artigo está explicada na Figura 9 como um diagrama de blocos.

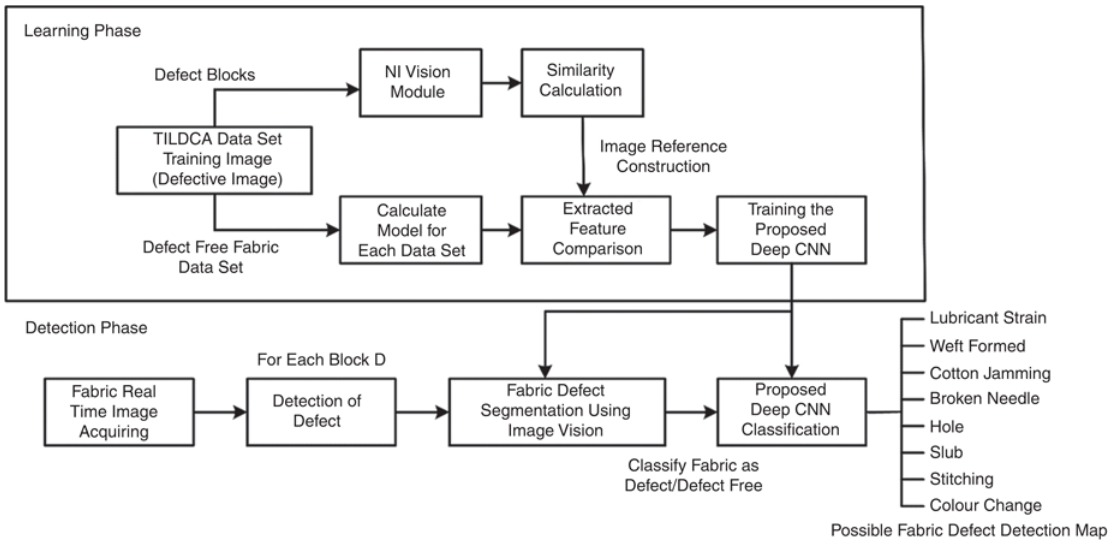


Figura 9: Representação em diagrama de blocos do sistema geral para classificação de defeitos. Imagem adaptada de Jeyaraj and Samuel Nadar [2019].

Posto isto, Jeyaraj and Samuel Nadar Jeyaraj and Samuel Nadar [2019] desenvolveram um sistema eficiente

e automatizado de detecção e classificação de defeitos em tecido baseado em visão por computador. Este modelo de protótipo foi testado em plataforma industrial em tempo real. O sistema desenvolvido é baseado em dois módulos: fase de aprendizagem de tecidos defeituosos (off) e fase de teste de materiais com tecidos defeituosos (tempo real). Na fase de teste o algoritmo foi avaliado recorrendo ao *dataset* padrão TILDA e em amostras de tecidos adquiridas em tempo real.

Em suma, para validar numericamente a eficácia do modelo CNN, este foi comparado com outras três abordagens comummente usadas na indústria moderna. Esta tabela de comparação entre modelos pode ser analisada na Figura 10. A partir dos resultados da detecção de defeitos, conclui-se que o algoritmo CNN proposto deteta a maioria dos defeitos de tecido, localizando ainda a região de defeito.

Autor, ano	Algoritmo usado para classificação de defeitos de tecido	Precisão (%)	Sensibilidade (%)	% De erro médio	Taxa de sucesso
Método proposto	Esquema de média de várias escalas na CNN	*96,55*	*96,4*	3.2.	*0,94*
Alawad e Lin (2016)	SVM	90,4	88,4	*2.7*	0,88
Hanbay *et al.* (2016)	Filtro de Gabor	91,8	87,5	4,15	0,92
Daniel *et al.* (2018)	CNN	94,5	96,2	3,82	0,81

Figura 10: Comparação do desempenho do modelo CNN com outros modelos de classificação de defeitos de tecido. Imagem adaptada de Hanbay et al. [2016].

2.3.3 *Comparação de diferentes metodologias na inspecção RGB*

O documento referenciado por Hanbay el al. Hanbay et al. [2016], aborda um trabalho de revisão abrangente da literatura sobre métodos de deteção de defeitos em tecido. Um defeito no tecido corresponde a uma falha na superfície do tecido fabricado. Segundo o estudo existem mais de 70 tipos de defeitos definidos pela industria, mas na generalidade estes podem ser classificados em dois que são: alteração da cor da superfície e irregularidade da textura local. Exemplos de defeitos frequentemente encontrados na industria estão na Figura 11.

Um sistema de detecção de defeitos de tecido melhora a qualidade do produto. Como resultado, os sistemas automatizados de detecção de defeitos de tecido para fabricar produtos de alta qualidade na industria têxtil estão em constante crescimento. Um sistema automatizado tem como função principal a identificação de falhas na superfície do tecido, usando as técnicas de processamento de imagem e/ou vídeo.

Os sistemas de inspecção de defeitos conseguem uma correção quase instantânea de pequenos defeitos, enquanto que a inspecção humana poderia falhar mais frequentemente devido a descuidos, ilusão óptica e cansaço por exemplo. Na inspecção automatizada, a deteção de defeitos é realizada em tempo real, estes

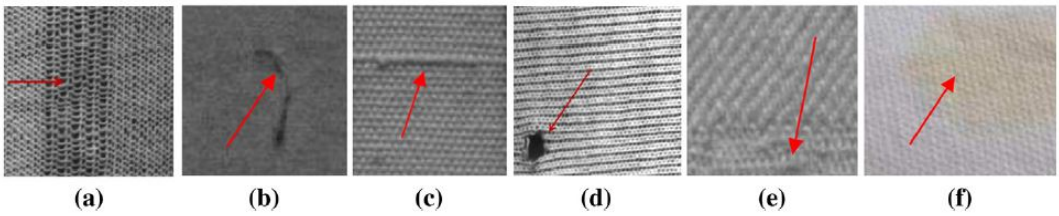


Figura 11: (a) quebra da agulha, (b) ondulação da trama, (c) barra, (d) orifício, (e) costura (f) manchas de ferrugem. Imagem adaptada de Hanbay et al. [2016].

sistemas detetam o defeito e interrompem o processo de produção só em caso de defeito.
Segundo este documento um sistema de deteção de defeitos de tecido é composto por três componentes: seleção da câmara, seleção da lente e por fim seleção da fonte de luz/iluminação.
A seleção da câmara pode ser exigente. Geralmente nestes tipos de sistemas usam-se dois tipos de câmaras, *area scan camera* e *line scan camera*. Na *area scan camera* a análise é feita com o tecido estático, enquanto que com a *line scan camera* a análise do tecido pode ser feita com este em movimento, o que torna esta segunda opção muito mais viável para uma linha de inspecção industrial.
Após a seleção da câmara adequada, é necessária uma seleção apropriada de lente. A área de inspecção e o campo de visão com uma câmara depende da lente usada. Portanto, a lente mais correta deve ser escolhida considerando valores como a distância de trabalho, o campo de visão e o tamanho do sensor.
Por último, a iluminação é considerada um problema fulcral nos sistemas de visão. Segundo o estudo existem quatro esquemas de iluminação usados nos sistemas de controlo automático de tecido, sendo estes: iluminação frontal, traseira, de fibra óptica e estruturada. Nestas aplicações, geralmente são usadas ou lâmpadas fluorescentes, ou iluminação halogena ou ainda, diferentes projetores de fontes de luz de díodos emissores de luz (LED).
Contudo o autor conclui, que não existe nenhum método capaz de ser aplicado a todos os tipos de tecido e que tivesse alta taxa de sucesso na deteção de todos os defeitos.
Como resultado, são apresentadas 3 figuras que comparam os métodos analisados no artigo: (Figura 12) métodos estatísticos, (Figura 13) métodos espectrais, e (Figura 14) métodos de ML.
As abordagens baseadas em ML apresentam resultados com maior precisão na deteção de defeitos em tecido, pois examinam as relações locais e globais dos pixeis.

2.4 Deep learning para classificação de imagens

A classificação de imagens é a tarefa de atribuir um rótulo a uma imagem de entrada a partir de um conjunto fixo de categorias. Esse é um dos principais problemas do *Computer Vision* e tem uma grande variedade de aplicações práticas. Esta tarefa de classificação pode parecer trivial mas enfrenta inúmeros desafios, como:

- **Variação do ponto de vista** - Um objeto pode ser orientado de várias maneiras em relação à câmara.
- **Variação de escala** - Um objeto pode assumir escalas enganosas numa imagem, o que pode dificultar a classificação.

Métodos	Pontos Fortes	Pontos Fracos
Gray level Co-occurrence matrix	➢ Extraindo relação espacial de pixéis com 14 cálculos estatísticos diferentes. ➢ Alta taxa de precisão.	➢ Computacionalmente caro para um sistema de inspeção de defeito em tempo real. ➢ Difícil determinar o melhor vetor de deslocamento. ➢ Exige procedimento de seleção de recurso. ➢ Depende da rotação e da escala.
Métodos com Histogramas	➢ Simplicidade computacional. ➢ Invariante para translação e rotação. ➢ Ideal para uso em aplicações em que se discrimina a tonalidade.	➢ Sensível ao ruído. ➢ Baixa taxa de deteção no erro de deteção de texturas não regulares.
Função Auto correlação	➢ Durável à alteração de ruído e iluminação. ➢ Complexidade de computação é bastante baixa. ➢ Realize uma medida direta e precisa da similaridade entre duas imagens. ➢ Adequado para tecidos lisos.	➢ Intensivo em computação para aplicações em tempo real e imagens de grande dimensão. ➢ Esta função pode ser sensível a interferência de ruído. ➢ Inadequado para tecido com texturas aleatórias.
Morfologia Matemática	➢ Eficiente em defeitos de imagem aperiódica. ➢ Representação geométrica da textura. ➢ Simplicidade computacional. ➢ Muito apropriado para texturas aleatórias ou naturais. ➢ Utilizável em máquinas de tecelagem e tricô para detetar e identificar defeitos.	➢ As operações morfológicas são implementadas apenas em defeitos de tecido não periódicos.

Figura 12: Pontos fortes e fracos na detecção de defeitos em tecido, usando métodos de abordagem estatística para tipos de tecidos e defeitos bem conhecidos.

Métodos	Pontos Fortes	Pontos Fracos
Transformada *Wavelet*	➢ Fornece análise de imagem em várias escalas. ➢ Identifica diferentes tipos de defeitos com diferentes *wavelets* mãe. Comprime imagens. ➢ Fornece alta taxa de precisão. ➢ Reduz ruído, alta taxa de precisão.	➢ Em uso adaptativo, alto custo computacional. ➢ Ele sofre interferência dos componentes da imagem ou apresenta correlações entre as escalas.
Transformada de *Fourier*	➢ O espectro de frequência espacial é invariante ao deslocamento, rotação e escala. ➢ Imagens de tecido são caracterizadas no domínio da frequência. ➢ Computação rápida e de fácil aplicação. ➢ Adequado na deteção de problemas globais defeitos locais (tecelagem e tricô).	➢ Não é possível detetar a textura aleatória da tela padronizada. ➢ Não funciona bem para deteção de defeitos em texturas aleatórias. ➢ Não é possível localizar as regiões defeituosas no domínio espacial.
Transformada de *Gabor*	➢ Oferece deteção ideal de defeitos para o domínio espacial e de frequência. ➢ Graças às diferentes escalas, oferece um espaço de recurso de alta dimensão. ➢ Um método de seleção de filtro adaptável é implementado para reduzir a complexidade computacional. ➢ Taxas de deteção perfeitas para defeitos com arestas e furos. ➢ Utilizável em tecelagem e tricô máquinas para detetar e identificar defeitos.	➢ A escolha de parâmetros ideais de filtro é bastante difícil. ➢ Não é invariável à rotação. ➢ Computação intensiva.
Abordagem de filtragem	➢ Uso intensivo em sistemas com métodos baseados em texto.	➢ Computação intensiva. ➢ Escolha de parâmetros ideias para o filtro é bastante difícil.

Figura 13: Pontos fortes e fracos na detecção de defeitos em tecido, usando métodos de abordagem espectral para tipos de tecidos e defeitos bem conhecidos .

- **Deformação** - Muitos objetos de interesse não são corpos rígidos e podem ser deformados, o que pode ser uma grande dificuldade na hora da classificação.
- **Oclusão** - Os objetos de interesse podem estar ocluídos. Às vezes, apenas uma pequena parte de um objeto (com apenas alguns pixels) pode estar visível.
- **Condições de iluminação** - Os efeitos da iluminação são muito relevantes ao nível do pixel.

Métodos	Pontos Fortes	Pontos Fracos
Método autorregressivo	➢ Examinou a relação linear entre pixels. ➢ Computação rápida. ➢ O modelo circular autorregressivo é invariável à rotação. ➢ Adequado para imagens de tecido com variações de padrão estocástico	➢ Sensível à iluminação e ao ruído. ➢ Baixa taxa de deteção para imagens de tecido com tamanho grande e irregular.
Método de campo aleatório de *Gauss Markov*	➢ Pode ser usado com estatísticas e métodos espectrais. ➢ Graças ao recurso de isotropia, adequado para aplicativos de segmentação. ➢ Captura a orientação local da textura em formação. ➢ Útil para modelar texturas de tecido.	➢ Não é possível detetar pequenos defeitos. ➢ Insuficiente em termos de análise global de textura. ➢ Não é invariável à rotação e escala.
Abordagem de aprendizagem	➢ Capacidade de aprender relacionamentos. complexos não lineares de entrada e saída. ➢ Trabalho eficaz devido a diferentes métodos de treinamento. ➢ O desempenho em tempo real é altamente adequado para a aplicação industrial. ➢ Utilizável em tecelagem e tricô.	➢ Computação intensiva para tamanho grande ➢ vetor de recurso

Figura 14: Pontos fortes e fracos da detecção de defeitos em tecido de métodos baseados em modelos de ML para tipos de tecido e defeitos bem conhecidos.

O objetivo do processo de classificação de uma imagem pode ser visto na Figura 15, onde uma imagem com *labeling* entra numa Rede Neuronal ajustada para a aplicação em causa e esta rede aprende os atributos dessa imagem. O objetivo é que ao fim dessa aprendizagem a rede consiga reconhecer todo o tipo de imagens (imagens totalmente desconhecidas), atingindo assim o objetivo para o qual se preparou.

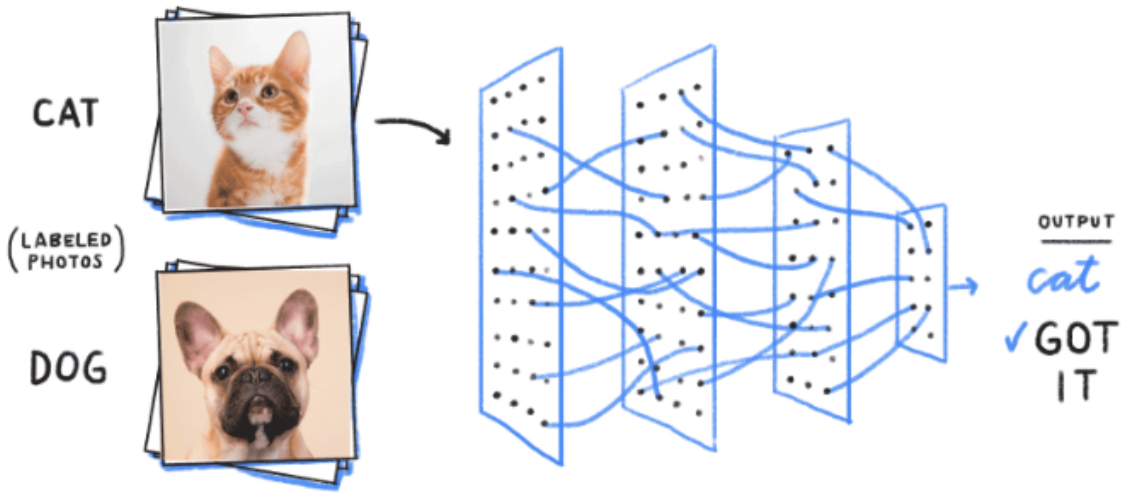


Figura 15: Ilustração do processo de classificação de uma imagem. Imagem adaptada de Sagar [2019].

2.4.1 *Convolutional neural network*

Quando se fala em visão computacional, uma das técnicas de DL mais famosa é CNN. As CNNs tornaram-se no principal método para resolver desafios que envolvam dados de imagens, quaisquer dados que tenham relações espaciais estão prontos para a aplicação da CNN. Quando nos deparamos com imagens de grandes resoluções o custo computacional e a consequente ocupação de memória está inerente, o uso de CNN consegue contornar estas desvantagens e gerar bons resultados a este nível.
As CNNs é uma das principais categorias na tarefa de reconhecimento e classificação de imagens. As classificações de imagem da CNN pegam numa imagem de entrada, processam-na e procedem à sua classificação em determinadas categorias. O computador vê uma imagem como uma matriz de pixeis, esta matriz contém uma altura, largura e dimensão. Por exemplo, uma imagem de matriz 6x6x3 (em que 6x6 diz respeito à dimensão e o 3 refere-se a valores RGB) e uma imagem de matriz 4x4x1, em que o 1 significa uma imagem em escala de cinza. Nos modelos de CNN cada imagem passa por uma serie de camadas de convolução com filtros (Kernels), Pooling, *Fully Connected Layer (FCL)* e aplica a função *Softmax* para classificar um objeto com valores probabilísticos entre 0 e 1. A representação do fluxo completo numa CNN está representado na Figura 16.

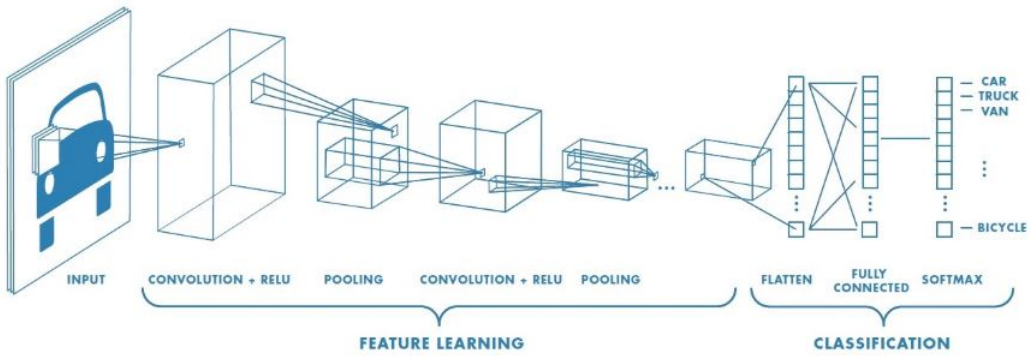


Figura 16: Representação do fluxo completo numa CNN para processar uma imagem de entrada e classificar os objetos de saída com base em valores. Imagem adaptada de Raghav [2018].

A convolução é a primeira camada a extrair recursos de uma imagem de entrada. Esta camada, denominada convolução preserva o relacionamento entre os pixeis, e traduz-se numa operação matemática que recebe duas entradas: sendo uma a matriz que representa a imagem e a outra um filtro (Kernel).

- Uma imagem com volume e dimensão sendo a altura, l largura e d a dimensão.

$$(a \times l \times d)$$

- Um filtro com volume e dimensão sendo fa altura do filtro, fl largura do filtro e fd a dimensão do filtro.

$$(fa \times fl \times fd)$$

- O *output* resulta da multiplicação da imagem de entrada pelo filtro e resulta em:

$$(a - fa + 1) \times (l - fl + 1) \times 1$$

Strides são os números de pixeis deslocados sobre a matriz de entrada. Quando o filtro não se encaixa perfeitamente na imagem de entrada, existem duas formas de contornar o processo, a primeira é colocar a imagem com zeros, a segunda é descartar a parte da imagem onde o filtro não coube. Quando se aumenta uma imagem de 5x5x1 em uma imagem 6x6x1 e, em seguida, se aplica um *Kernel* 3x3x1 sobre a imagem aumentada, observa-se que a matriz resultante apresenta dimensões 5x5x1. Daí o nome *same padding*. Por outro lado, se a mesma operação for realizada mas desta vez sem preenchimento, a matriz resultante apresenta as dimensões do próprio *Kernel* (3x3x1), denominando-se assim *valid padding*, Raghav [2018].

Outro conceito importante é o de *Pooling Layer*, a sua função é reduzir o tamanho espacial do recurso envolvido (redução dimensional), desta forma reduz a potência computacional necessária para processar os dados. A *Pooling Layer* é muito útil na extração de características dominantes que não variam tornando assim o processo igualmente eficaz.
Existem dois processos de *Pooling*: *Max Pooling* e *Average Pooling*, na Figura 17 os processos estão explicados de uma forma gráfica. *Max Pooling* retorna o valor máximo da imagem coberta pelo *Kernel*, por outro lado, o *Average Pooling* retorna a média de todos os valores da parte da imagem coberta pelo *Kernel*.

- *Max Pooling* - funciona como um supressor de ruído, pois descarta completamente as ativações ruidosas e realiza a remoção do ruído junto com a redução da dimensão.

- *Average Pooling* - realiza apenas uma redução da dimensão.

Portanto, pode-se afirmar que o *Max Pooling* tem um desempenho muito melhor em comparação com o *Average Pooling*, Raghav [2018]. ReLU significa Unidade Linear Retificada para uma operação não linear. A

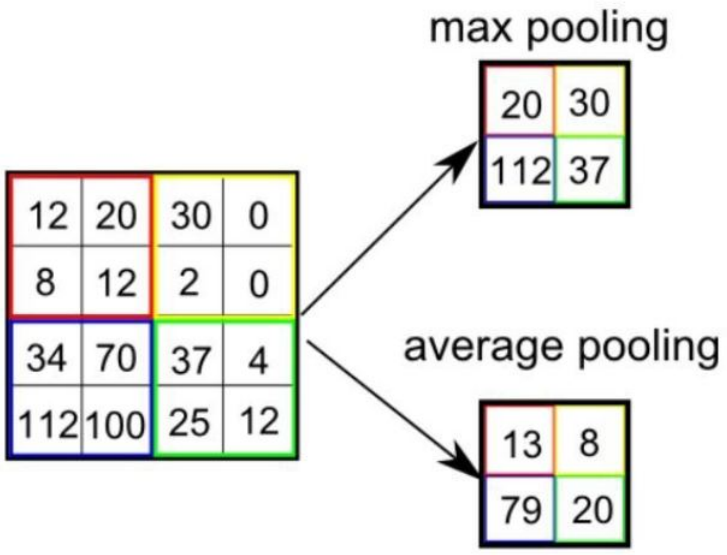


Figura 17: Ilustração exemplificativa do tipos de *Pooling*. Imagem adaptada de Saha [2018].

saída da função é dada por:

$$f(x) = max(0, x) \tag{1}$$

O objetivo da ReLU é introduzir a não linearidade. Existem outras funções não lineares, como *tanh* ou sigmóide, que também podem ser usadas no lugar da ReLU. Mas a função ReLU é a mais usada, porque apresenta um desempenho superior a outras existentes, Saha [2018].

FCL é outro conceito importante em redes neuronais. Estamos perante uma FCL quanto as entradas de uma camada são conectadas a todas as unidades de activação da camada seguinte. Nos modelos mais populares de ML, as últimas camadas são FCL que compilam os dados extraídos pelas camadas anteriores formando assim a saída final. A segunda camada é a que consome mais tempo, ver esquema na Figura 18.

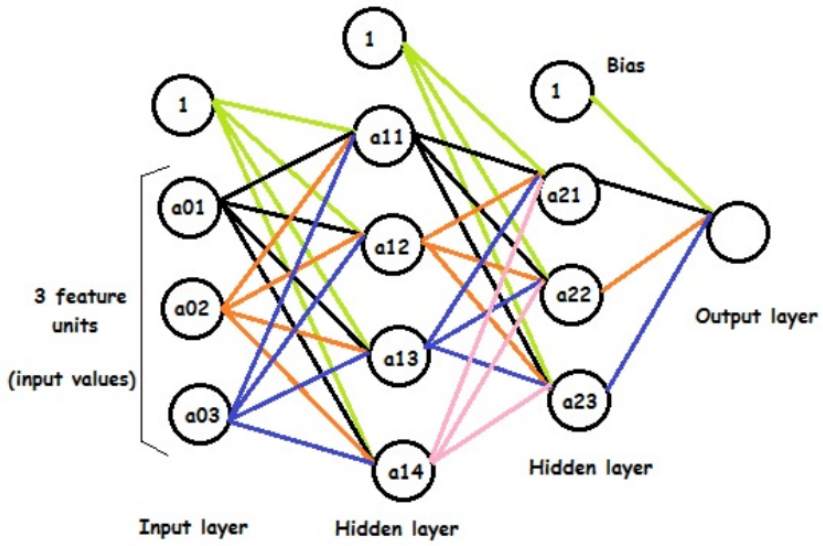


Figura 18: Rede Neuronal com *Fully Connected layers*. Imagem adaptada de Singh [2017].

Concluindo, uma CNN é um algoritmo de DL que pode reconhecer e classificar *features* em imagens para visão computacional. É uma rede neuronal de várias camadas, projectada para analisar entradas visuais e executar tarefas como classificação de imagens, segmentação e detecção de objetos, que podem ser úteis na condução autónoma. As CNNs podem também ser usadas na aplicação de DL na área da saúde, como imagens médicas. Uma CNN é composta por duas partes: a convolução que divide as várias *features* da imagem para análise, e ainda a existência de uma FCL que usa a saída da camada de convolução para prever a melhor descrição para a imagem.

A arquitetura da CNN é inspirada na organização e na funcionalidade do córtex visual e projectada para imitar o padrão de conectividade dos neurónios no cérebro humano.

Os neurónios dentro de uma CNN são divididos numa estrutura tridimensional, em que cada conjunto de neurónios analisa uma pequena região ou característica da imagem. Ou seja, cada grupo de neurónios é especializado em identificar uma parte da imagem. As CNNs usam as previsões das camadas para produzir uma saída final que apresenta um vector de pontuações de probabilidade que representa a probabilidade de um recurso específico pertencer a uma determinada classe.
Na Figura 19 está representada a arquitetura de uma CNN. Em suma, uma CNN é composta pelas seguintes camadas:

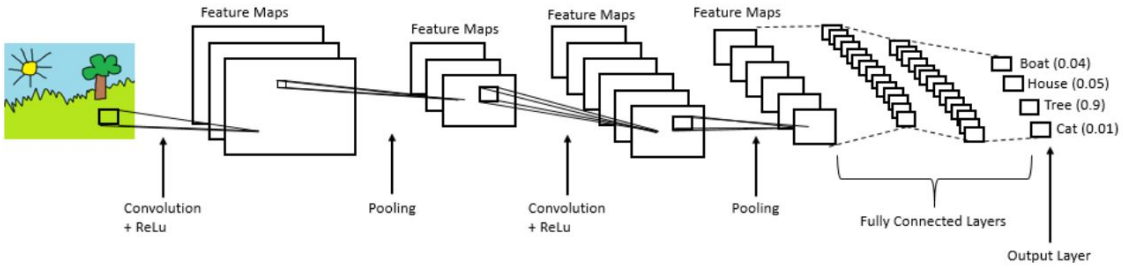


Figura 19: Arquitectura completa da CNN. Imagem adaptada de Saha [2018].

- Camada convolucional - esta camada cria um *feature map* que prevê as probabilidades das classe para cada *feature*, com a aplicação de um filtro que passa pela imagem inteira através de poucos pixeis de cada vez.
- Camada de pool (downsampling) - reduz a quantidade de informações que a camada convolucional gerou para cada *feature* mantendo as informações essenciais (o processo das camadas convolucionais e de pool geralmente repetem-se várias vezes).
- *Fully connected input layer* - "nivela" as saídas geradas pelas camadas anteriores transformando-as num vetor único que pode ser usado como entrada para a próxima camada.
- *Fully connected layer* - aplica pesos sobre a entrada gerada pela análise de recursos para prever uma *label* precisa.
- *Fully connected output layer* - gera as probabilidades finais para determinar uma classe para a imagem.

2.4.2 *Arquiteturas de CNNs*

Existem várias arquiteturas de CNNs disponíveis, ver Figura 20, como: LeNet-5, AlexNet, VGG-16, Inception-v1, Inception-v3, ResNet-50, Xception, Inception-v4, Resception-ResNets e ResNeXt-50. Estas redes tornaram-se tão profundas e complexas que se tornou extremamente difícil visualizar o modelo inteiro. Devido à sua complexidade, começaram a ser tratadas como modelos caixa preta, Karim [2019].
A arquitetura de uma CNN é o fator chave para determinar o seu desempenho e eficiência. A forma como as camadas estão estruturadas, quais elementos são usados em cada camada e como estes são projectados

afectam frequentemente a velocidade e a precisão com que ela pode executar várias tarefas.

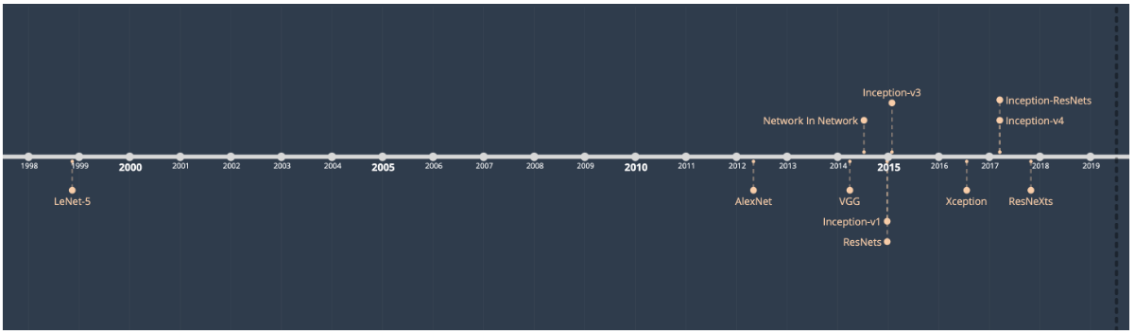


Figura 20: Evolução cronológica do aparecimento de novas arquiteturas CNN. Imagem adaptada de Karim [2019].

Existem várias arquiteturas populares de CNN, ver Figura 20, muitas delas ganharam reconhecimento ao obter bons resultados no *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), segundo a plataforma de DL Shenhav.

- LeNet-5 (1998) - É uma arquitetura CNN de 7 camadas, ver Figura 21, que serviu em 1998 para classificar dígitos a partir de imagens de entrada digitalizadas em escala de cinza de 32 × 32 pixeis. Esta arquitetura foi usada por vários bancos para reconhecer os números escritos à mão nos cheques.

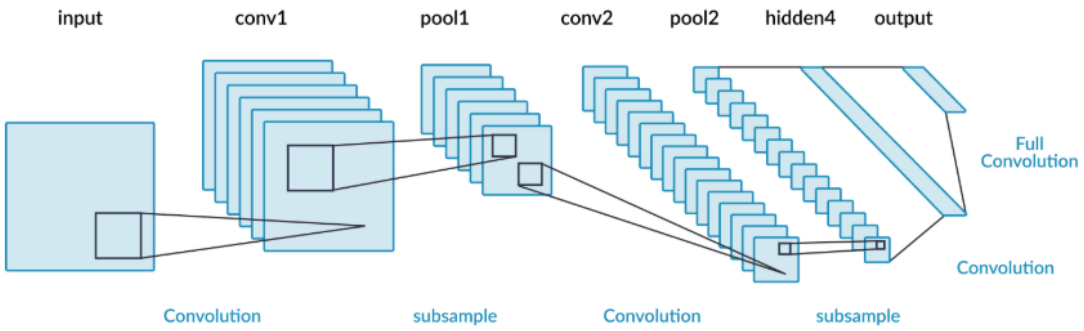


Figura 21: Arquitectura LeNet-5. Imagem adaptada de Shenhav.

- AlexNet (2012) - A AlexNet é uma arquitetura projetada pelo grupo *SuperVision*, sendo esta semelhante à LeNet, mas mais profunda (Figura 22). Esta arquitetura apresenta mais filtros por camada, além de camadas convolucionais empilhadas. É composta por cinco camadas convolucionais, seguidas por três *fully connected layers*.
  Uma das diferenças mais significativas entre o AlexNet e outros algoritmos de detecção de objetos é o uso da ReLU para a parte não linear em vez do uso de funções Sigmond ou Tanh muito usadas em redes neuronais tradicionais. AlexNet aproveita o treino mais rápido da ReLU para tornar o seu algoritmo também mais rápido.

Os criadores da AlexNet dividiram a sua rede em dois *pipelines* porque usaram duas GPUs (Nvidia Geforce GTX 580 Graphics Processing Units) para treinar esta CNN.

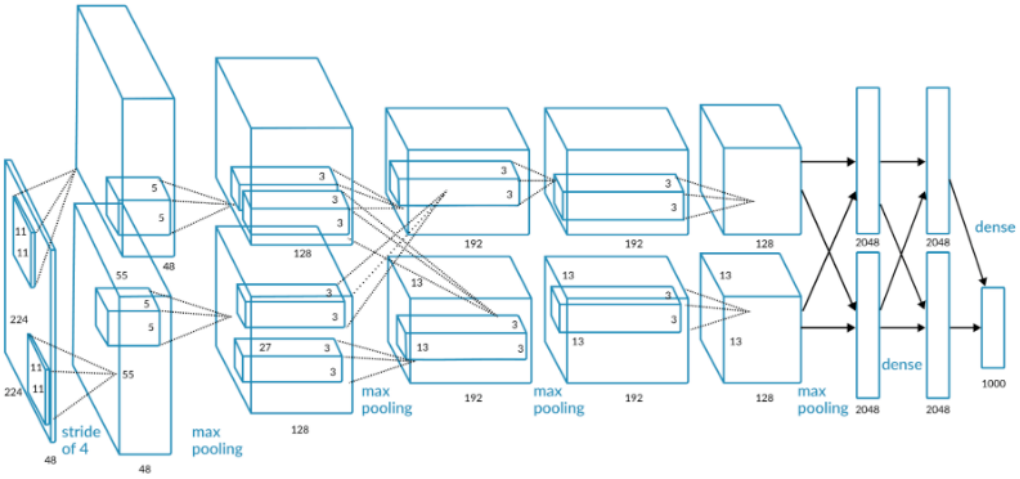


Figura 22: Arquitectura AlexNet. Imagem adaptada de Shenhav.

- GoogleNet (2014) - É uma criação inspirada na LetNet, esta rede também denominada de Inception V1, foi criada pela Google, ver esquema na Figura 23. O GoogleNet foi o vencedor do ILSVRC 2014 e alcançou uma taxa de erro mínima de 7%, percentagem próxima do nível de desempenho humano. A arquitetura do GoogleNet consiste numa CNN de 22 camadas de profundidade que usa um módulo baseado em pequenas convoluções, chamado "inception module", que usa batch normalization e RMSprop.

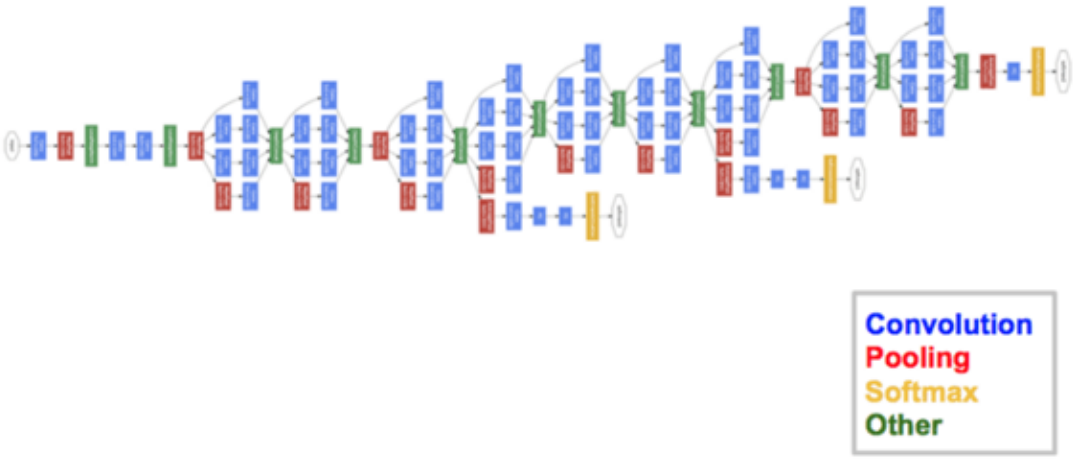


Figura 23: Arquitectura GoogleNet. Imagem adaptada de Shenhav.

- VGGNet (2014) - A VGGNet, foi a segunda rede com melhor desempenho apresentada no ILSVRC 2014, e possui 16 camadas convolucionais, ver Figura 24. A VGGNet foi treinada em 4 GPUs por mais de

duas semanas para alcançar o seu desempenho. O problema com o VGGNet são os 138 milhões de parâmetros, 34.5 vezes mais que o GoogleNet, o que dificulta a sua execução.

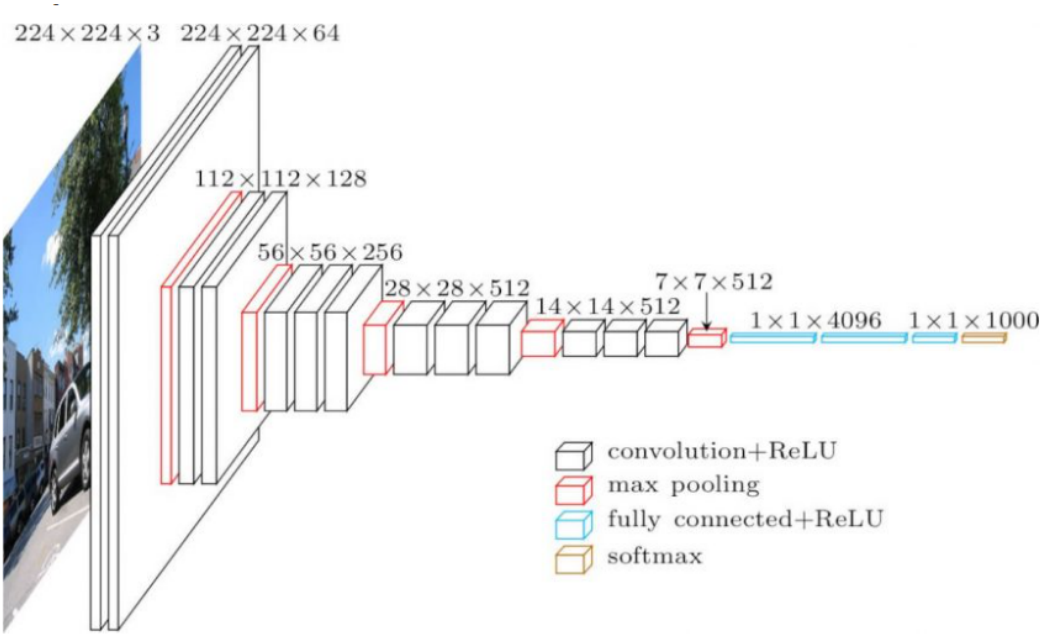


Figura 24: Arquitectura VGGNet. Imagem adaptada de ul Hassan [2018b].

2.4.3 *Métricas*

Num qualquer modelo de ML ou DL existe a necessidade de "medir"a qualidade do modelo de acordo com o objetivo da tarefa a que se propõe. Com o aparecimento da necessidade de avaliar estes modelos foram desenvolvidas funções matemáticas que ajudam na avaliação da capacidade de erro e no acerto dos modelos. Estas funções matemáticas denominam-se de métricas.

A importância de escolher o melhor modelo de ML ou DL para uma dada tarefa é indiscutível. Mas, é a partir da escolha da métrica apropriada ao problema que conseguimos avaliar o desempenho dos modelos e consequentemente compara-los entre si.

Existem vários tipos de métricas, umas mais simples outras mais complexas, umas comportam-se melhor consoante determinadas características de *datasets*, outras são desenvolvidas consoante o objetivo dos modelos. A escolha da métrica mais acertada pode fazer a diferença na hora da avaliação do modelo, por isso, devem-se considerar fatores como proporção de dados nos *dataset* e o objetivo da previsão como: probabilidade, binário, *ranking*, etc.

O conceito de métrica perfeita não existe e nem deve existir essa comparação, todas elas são igualmente relevantes, depende unicamente da aplicação prática do modelo em que são usadas. Em certas situações especificas o ideal é que a métrica usada tenha um significado especifico para a tarefa em questão, por exemplo imagine-se a situação de sites de anúncios, uma boa métrica é a taxa de cliques, com isto, é importante interiorizar que as métricas a seguir apresentadas podem ser usadas na generalidade mas se numa situação especifica se conseguir uma métrica diretamente relacionada com o contexto, é essa que deve ser usada.

As métricas a seguir apresentadas, são utilizadas em tarefas de classificação. A tarefa de classificação é procurar prever qual é a categoria a que uma dada amostra pertence. A maioria das métricas pode ser usada tanto na tarefa de classificação binária como em classificação de várias classes.

Matriz de confusão

A matriz de confusão é uma medida de desempenho para a classificação de ML em que a saída pode ser duas ou mais classes. É uma tabela com 4 combinações diferentes de valores previstos e reais, ver Tabela 1.

Tabela 1: Matriz Confusão.

		Valor Verdadeiro	
Valor Previsto		Positivos	Negativos
	Positivos	**VP** **Verdadeiro Positivo**	**FP** **Falso Positivo**
	Negativos	**FN** **Falso Negativo**	**VN** **Verdadeiro Negativo**

1. Verdadeiro Positivo (VP): Valores que pertencem à classe positiva e foram classificados como positivos. Classificados corretamente.
2. Falso Positivo (FP): Valores que pertencem à classe negativa e foram classificados como positivos.
3. Falso Negativo (FN): Valores que pertencem à classe positiva e foram classificados como negativos.
4. Verdadeiro Negativo (VN): Valores que pertencem à classe negativa e foram classificados como negativos. Classificados corretamente.

As métricas a seguir apresentadas são amplamente usadas na comparação de modelos de classificação. Cada uma das seguintes métricas avalia um aspecto diferente do modelo. Para perceber as métricas é importante ter em mente a Matriz de Confusão apresentada anteriormente na tabela 1.

1. *Accuracy*: Corresponde à proporção de casos verdadeiros (Verdadeiro Positivo e Verdadeiro Negativo) entre o número total de casos examinados.

$$Accuracy = \frac{VerdadeiroPositivo(VP) + VerdadeiroNegativo(VN)}{Total} \tag{2}$$

2. *Precision*: Número de exemplos classificados como pertencentes a uma classe, que realmente são daquela classe (Verdadeiros Positivos), dividido pela soma entre este número, e o número de exemplos classificados nesta classe, mas que pertencem a outras (Falsos Positivos).

$$Precision = \frac{VerdadeirosPositivos(VP)}{VerdadeirosPositivos(VP) + FalsosPositivos(FP)} \tag{3}$$

3. *Recall*: Frequência com que o classificador encontra os exemplos de uma classe, ou seja, quando realmente é da classe x, o quanto frequentemente é classificado como x:

$$Recall = \frac{VerdadeiroPositivo(VP)}{VerdadeiroPositivo(VP) + FalsoNegativo(FN)} \tag{4}$$

4. *F1 Score*: Métrica que combina *precision* e *recall* de modo a traduzir num único valor a qualidade geral do modelo, esta métrica trabalha bem com conjuntos de dados que possuam classes desproporcionais. Quanto maior o valor, melhor o modelo.

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{5}$$

5. *Intersection over Union* (IoU): No problema de deteção de objetos esta é uma métrica muito utilizada e bastante eficiente na análise do problema. Na Figura 25 está ilustrada a equação que traduz a métrica IoU, este calculo consiste em dividir a área de interceção entre a localização do objeto e a previsão deste pela área de união (objeto e previsão). Na Figura 26, consegue-se perceber e interpretar esta métrica através do valor de IoU.

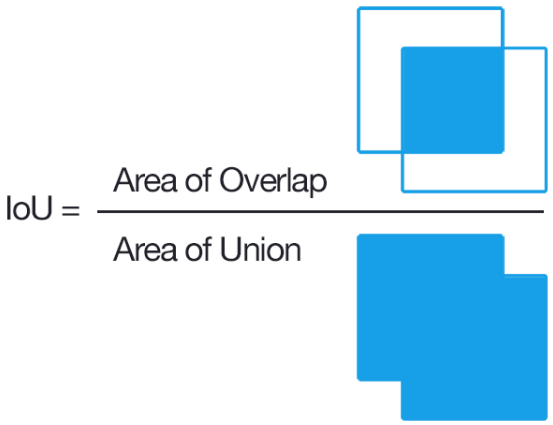
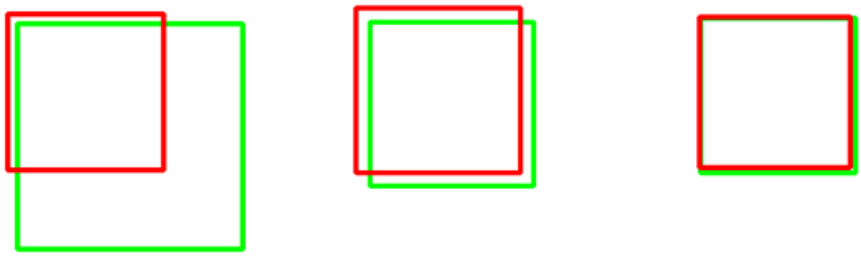


Figura 25: Representação da equação que traduz o valor de IoU. Imagem adaptada de Rosebrock [2016].

2.5 Object detection

A OD é uma tecnologia relacionada com visão computacional e com o processamento de imagens, esta tecnologia consiste na deteção de objetos semânticos de uma determinada classe, como por exemplo, pessoas, casas ou carros. Esta deteção é feita através de imagens ou vídeos.
Esta tecnologia é muito utilizada em tarefas de visão computacional. A OD consiste em atribuir a cada objeto uma classe, sendo que cada classe possui certos recursos que ajudam na sua classificação, por exemplo, os círculos são redondos, logo o OD sabe que está perante um círculo quando é encontrado um objeto que tem uma distância especifica de um ponto (centro do círculo).
Os métodos de OD enquadram-se geralmente nas abordagens de ML e DL. Nas abordagens de ML é necessário definir primeiramente recursos usando um dos seguintes métodos:



(a) (b) (c)

Figura 26: (a) Valor de IoU = 0.4034 (Mau), (b) Valor de IoU = 0.7330 (Bom) (c) Valor de IoU = 0.9264 (Excelente). Imagem adaptada de Rosebrock [2016].

- *Viola–Jones Object Detection framework based on Haar features*;
- *Scale-invariant feature transform (SIFT)*;
- *Histogram of oriented gradients (HOG) features*.

Posteriormente é usado *Support Vector Machine (SVM)* na tarefa de classificação. Por outro lado, na abordagem DL que normalmente são baseadas em CNN é possível detetar objetos sem definir recursos. As abordagens de DL são:

- *Region Proposals (R-CNN, Fast R-CNN, Faster R-CNN)*;
- *Single Shot MultiBox Detector (SSD)*;
- *You Only Look Once (YOLO)*;
- *Single-Shot Refinement Neural Network for Object Detection (RefineDet)*.

Em seguida, será apresentado por ordem cronológica o estudo de alguns dos métodos de DL mais famosos.

2.5.1 *Region - based Convolution Neural Networks - 2014*

Region - based Convolution Neural (R-CNN) é um método de OD apresentado por Girshick et al. [2014] em outubro de 2014. O R-CNN é composto por duas etapas:

- A primeira consiste em usar *Selective Search (SS)*, esta etapa consiste em identificar regiões candidatas a detentoras de objetos ROI.
- Na segunda e última etapa, as propostas de ROIs são as *features* que entram na CNN e que serão individualmente classificadas.

Na Figura 27 é apresentada a arquitetura do R-CNN.

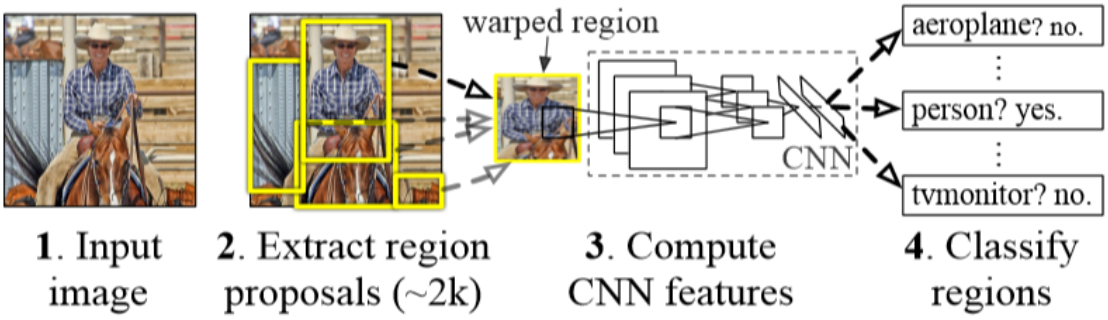


Figura 27: Arquitetura do método de *Object Detection* R-CNN. Imagem adaptada de Girshick et al. [2014]. (1) exemplo da imagem que será classificada pelo método R-CNN. (2) esta imagem é divida em regiões de interesse (possíveis regiões detentoras de objetos com interesse para classificação) através de um algoritmo denominado SS, as possíveis regiões de interesse que resultaram deste algoritmo estão discriminadas com caixas de cor amarela. (3) as regiões resultantes do SS, são os inputs da CNN, está arquitetura requer um *input* de 227x227, sendo por isso possível observar uma das ROI da figura em *warped region*. (4) por último, cada uma das ROI é classificada usando um classificador especifico linear SVM.

A abordagem de SS é aplicada com o intuito de encontrar ROIs para classificação, esta abordagem consegue capturar muitas possibilidades em diferentes escalas com a vantagem de fazer esta seleção requerendo baixa complexidade computacional. São seleccionadas 2000 regiões em cada imagem, Weng [2017]. As vantagens do SS é conseguir encontrar regiões de todas as escalas, ou seja, ter bastante diversificação e conseguir o cálculo destas regiões rapidamente.

Inicialmente, é aplicado um algoritmo de segmentação de imagem baseado em gráfico, Felzenszwalb and Huttenlocher [2004] para começar a criar regiões. O autor aborda o problema da segmentação de uma imagem em regiões. Uma característica importante do método é a capacidade de preservar detalhes em regiões da imagem de baixa variabilidade enquanto ignora detalhes em regiões de alta variabilidade. Na Figura 28 é possível perceber a aplicação deste algoritmo.

Figura 28: Resultado da aplicação do método *Efficient Graph-Based Image Segmentation*. Imagem adaptada de Felzenszwalb and Huttenlocher [2004].

Seguidamente, é usado um algoritmo para agrupar regiões de forma iterativa, que pode ser descrito em duas etapas:

- Inicialmente são calculadas todas as regiões vizinhas.

- Depois de duas regiões semelhantes serem agrupadas, novas semelhanças são calculadas entre a região calculada e os seus vizinhos.

Este processo de agrupamento das regiões mais semelhantes é repetido até que passe pela a totalidade da imagem.

O processo de implementação da R-CNN requer os seguintes passos:

1. É necessário pré-treinar uma rede CNN em tarefas de classificação de imagens como por exemplo: VGG ou ResNet.

2. São propostas regiões de interesse independentes da categoria por SS, o que se traduz em cerca de 2000 candidatos de região por imagem. Estas regiões podem conter objetos de interesse e podem ser de diversos tamanhos.

3. As ROIs calculadas são redimensionadas para um tamanho fixo, conforme exigido pela CNN.

4. Muitas das classes classificadas pela CNN tratam-se de *background*, ou seja, nenhum objeto de interesse. Numa fase inicial deve-se começar com um *learning rate* baixo.

5. Cada região da imagem gera um *feature vector*. Este é consumido por um support vector machine binário treinado para cada classe independentemente. As amostras positivas são regiões propostas de sobreposição de IoU maior ou igual que 0.3.

6. Por último, para reduzir erros de localização, um modelo de regressão é treinado usando CNN *features*, para gerar coordenadas mais ajustadas para as caixas de classificação do objeto depois deste já ter sido classificado.

Observando as etapas acima da R-CNN, facilmente se percebe que este é um modelo caro a nível computacional o que o torna muito lento. Não podendo ser implementado em tempo real, pois leva cerca de 47 segundos para a classificação de uma só imagem, ul Hassan [2018a].

2.5.2 *Fast Region - based Convolution Neural Networks - 2015*

Fast Region - based Convolution Neural Networks (Fast R-CNN) é um método de OD apresentado por Girshick [2015] em setembro de 2015. O objetivo da criação deste modelo de OD é melhorar a questão do tempo no modelo apresentado na secção 2.5.1. Então, de forma a acelerar a R-CNN o autor Girshick [2015] aprimorou o procedimento de treino. A abordagem é semelhante à R-CNN, mas neste método em vez de alimentar a CNN com as ROIs usa-se a imagem completa como entrada da CNN, ver Figura 29. Uma CNN principal com várias camadas convolucionais está a receber a imagem inteira como entrada, em vez de usar uma CNN para cada ROI como na arquitetura R-CNN. A região de interesse ROI é detectada com o método de SS aplicado nos *feature maps* produzidos. Formalmente, o tamanho dos *feature maps* é reduzido usando uma camada de pool de ROI para obter ROIs válidas com altura e largura fixas como hiperparâmetros. Cada camada ROI alimenta FCL, criando *features vectores*. Estes *vectores* são usados para prever o objeto observado com um classificador *softmax* e adaptar as localizações das *bounding boxes* com um regressor linear. Estes passos estão ilustrados na Figura 30.

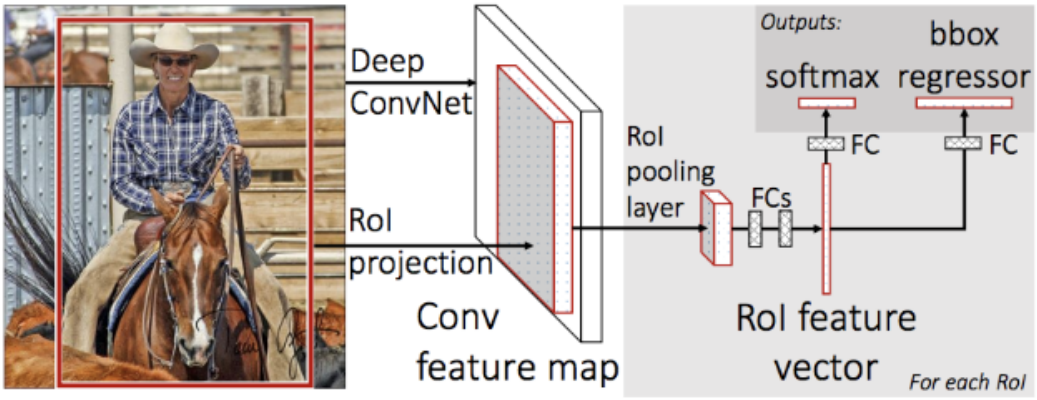


Figura 29: Representação da arquitetura do método de OD Fast R-CNN. Imagem adaptada de Girshick [2015]. Uma imagem de entrada e várias ROIs (regiões de interesse) são inseridas numa *fully convolutional network*. Cada ROI é agrupada num *feature map* de tamanho fixo e, em seguida, mapeada para um vetor de *features* por FCL. A rede possui dois vectores de saída por ROI: probabilidades softmax e compensações por regressão de *bounding-box* por classe. A arquitetura é treinada de ponta a ponta com um *multi-task loss*.

Os melhores ensaios usando o Fast R-CNNs atingiram pontuações mAP de 70.0% no conjunto de dados de teste PASCAL VOC de 2007, 68.8% no conjunto de dados de teste PASCAL VOC de 2010 e 68.4% no conjunto de dados de teste PASCAL VOC de 2012, Ouaknine [2018].

2.5.3 *Faster Region - based Convolution Neural Networks - 2016*

Faster R-CNN é um método de OD apresentado por Ren et al. [2016] em janeiro de 2016. Tornou-se um algoritmo muito popular e consequentemente atraiu a atenção de muitos cientistas de dados, cientistas de DL e engenheiros de inteligência artificial.
A arquitetura do Faster R-CNN é composta por 3 partes, Figura 31:

1. Camadas de Convolução (Figura 31 (a)) - O objetivo da existência destas camadas, é extrair as *features* mais relevantes nas imagens de entrada. Por exemplo, imagine-se que se quer extrair *features* para o rosto humano, os filtros têm como objetivo aprender através de formas e cores o rosto humano, para estes filtros só pode existir o rosto humano como resultado. O autor ul Hassan [2018a] serve-se de um exemplo para explicar estas camadas de convolução, comparando-as com um filtro de café, imagine-se uma chávena com um filtro de café em cima, o filtro de café pode ser comparado com as camadas de convolução, o filtro não pode deixar passar o pó do café para a chávena assim como, as camadas não podem deixar passar as *features* da imagem apenas o objeto desejado comparando este ao café (objeto de interesse como resultado deste processo).
    - Pó de café + liquido de café = Imagem de entrada
    - Filtro de café = filtros CNN
    - Café liquido = *Last feature map of the CNN*

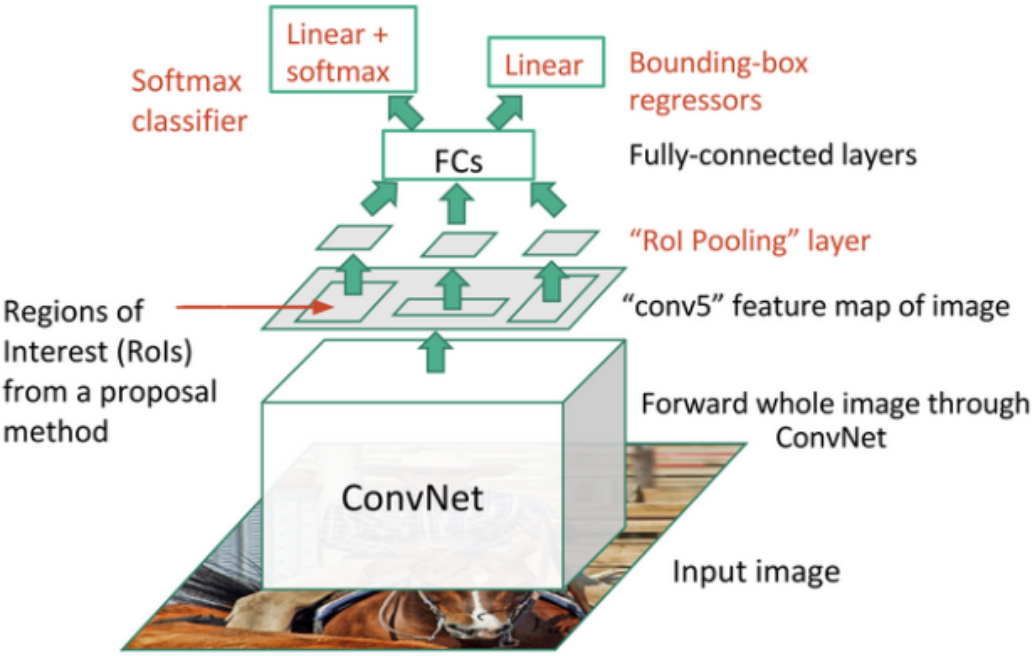


Figura 30: Arquitetura do método OD Fast R-CNN. Imagem adaptada de Ouaknine [2018].

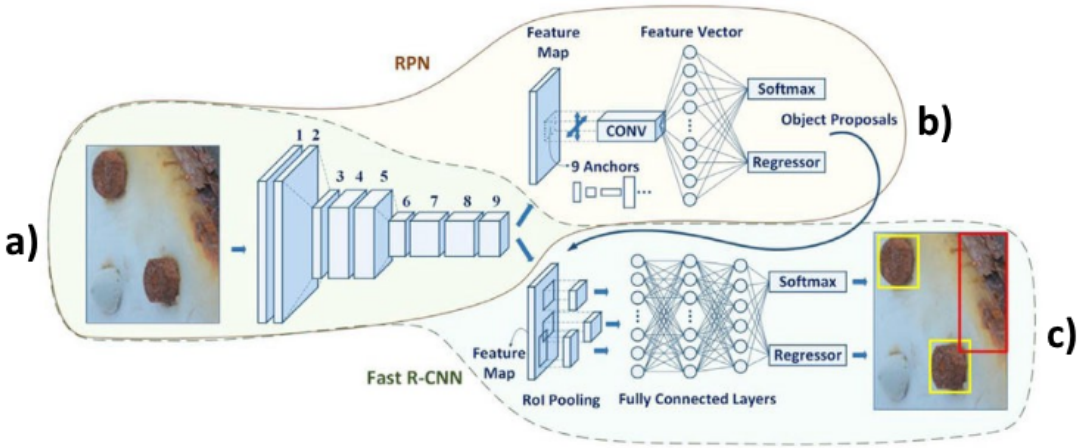


Figura 31: Representação da arquitetura do método OD Faster R-CNN. Imagem adaptada de KHAZRI [2019].

As redes de convolução são compostas por camadas de convolução, camadas de *pooling* e em último FCL ou outra qualquer extensão que será responsável pela classificação ou deteção.

A convolução é calculada deslizando o filtro ao longo de toda a imagem de entrada e o resultado é uma matriz de duas dimensões chamado *feature map*. O *pool* consiste em diminuir a quantidade de *features* no *feature map*, eliminando os pixeis com os valores mais baixos. No fim, é usada a FCL para classificar as *features*, ver Figura 32.

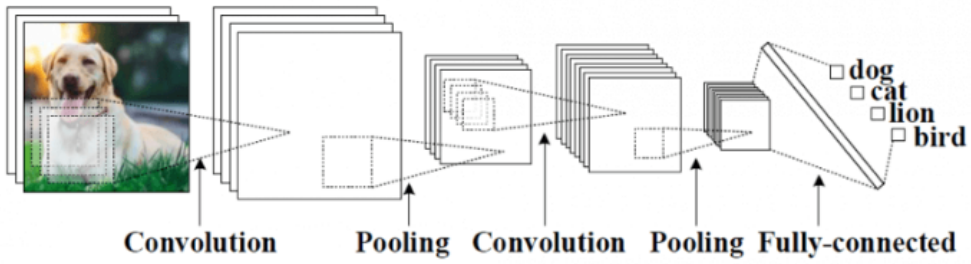


Figura 32: Esquema representativo de um Faster R-CNN. Imagem adaptada de KHAZRI [2019].

2. Region Proposel Network (RPN) - Uma RPN consiste numa pequena rede neuronal que desliza no último *feature map* das camadas de convolução e prevê se existe ou não um objeto, caso exista prevê também a *bounding boxes* desse objeto, ver Figura 31 (b)).
   O RPN possui uma arquitetura especializada, ver Figura 33. O RPN possui um classificador e um regressor. O classificador determina a probabilidade de uma proposta ter um objeto alvo. O regressor regride as coordenadas das propostas.

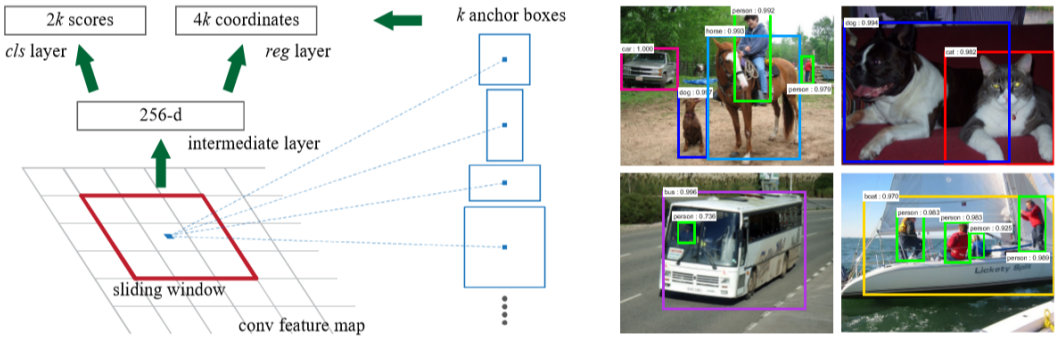


Figura 33: Esquema exemplificativo do algoritmo de *Region Proposal Network*. Imagem adaptada de Ren et al. [2016].

3. Previsão de classes e *bounding boxes* - A última etapa consiste em usar outras redes neuronais totalmente conectadas em que o *input* consiste nas regiões propostas pela RPN e prever assim a classe do objeto (classificação) e as *bounding boxes* (regressão).

Em suma, na Figura 34 está apresentado um gráfico em que é possível comparar o tempo que cada método R-CNN demora na inferência e uma imagem e é possível comprovar a eficiência deste método a nível de tempo, concluindo assim, que o Faster R-CNN pode ser usado na deteção de objetos em tempo real.

2.5.4 *Single Shot Detector - 2016*

Este modelo foi lançado no final de novembro de 2016 por Liu et al. [2015] e alcançou recordes em termos de desempenho e precisão para tarefas de OD, com mais de 74% de mAP a 59 FPS em *datasets* padrão

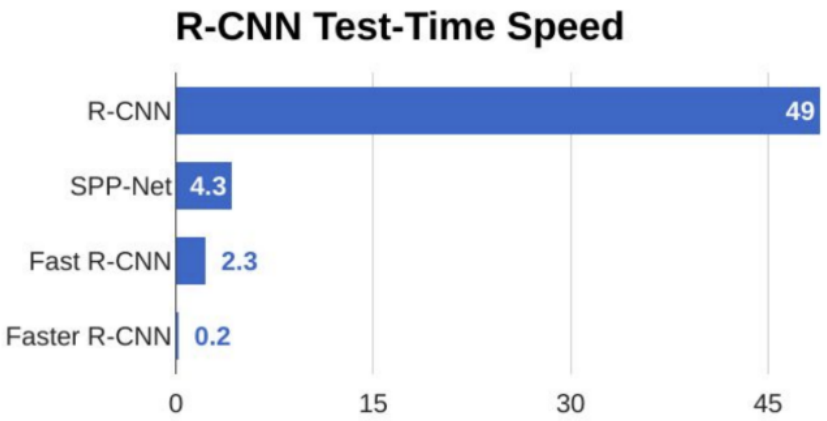


Figura 34: Comparação da velocidade na inferência dos principais métodos de OD. R-CNN secção 2.5.1, Fast R-CNN secção 2.5.2, Faster R-CNN secção 2.5.3. Imagem adaptada de Gandhi [2018].

como PascalVOC e COCO. O SSD foi projetado essencialmente para detecção de objetos em tempo real. SSD é um algoritmo de OD que consiste numa modificação da arquitetura VGG-16, descartando as FCL desta arquitetura.

Segundo Dash [2018], a principal vantagem que levam o autor do SSD a usar a VGG-16 como arquitetura base é o forte desempenho na tarefa de classificação de imagens de alta qualidade.

Em vez do uso de FCL originais do VGG-16, no SSD foi adicionado um conjunto de camadas convolucionais auxiliares (a partir de conv6), ver Figura 36. Permitindo assim, extrair *features* em várias escalas e diminuir progressivamente o tamanho da entrada para cada camada subsequente.

De acordo com a comparação feita pelo autor Hui [2018], o SSD atinge uma velocidade de processamento em tempo real que supera a precisão do conhecido método de OD Faster R-CNN, ver Figura 35. Para que o

System	VOC2007 test *mAP*	FPS (Titan X)	Number of Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	~6000	~1000 x 600
YOLO (customized)	63.4	45	98	448 x 448
SSD300* (VGG16)	77.2	46	8732	300 x 300
SSD512* (VGG16)	**79.8**	19	24564	512 x 512

Figura 35: Comparação do desempenho entre redes de deteção de objetos. Imagem adaptada de Hui [2018].

método SSD consiga atingir mais precisão na deteção, diferentes camadas dos *feature maps* estão a passar também por uma pequena convolução 3x3, ver Figura 36, Tsang [2018].

O SSD faz muitas previsões, 8732 *bounding boxes*, para uma melhor cobertura da localização, escala e proporção, tornando-se assim melhor do que muitos outros métodos de detecção. No entanto, muitas previsões não contêm objetos. Portanto, as previsões com pontuação de confiança de classe menor que 0.01 são eliminadas. A comparação da performance entre SSD e outros métodos de OD estão na Figura 37. Hui Hui [2018] apresenta as seguintes conclusões sobre o método SSD:

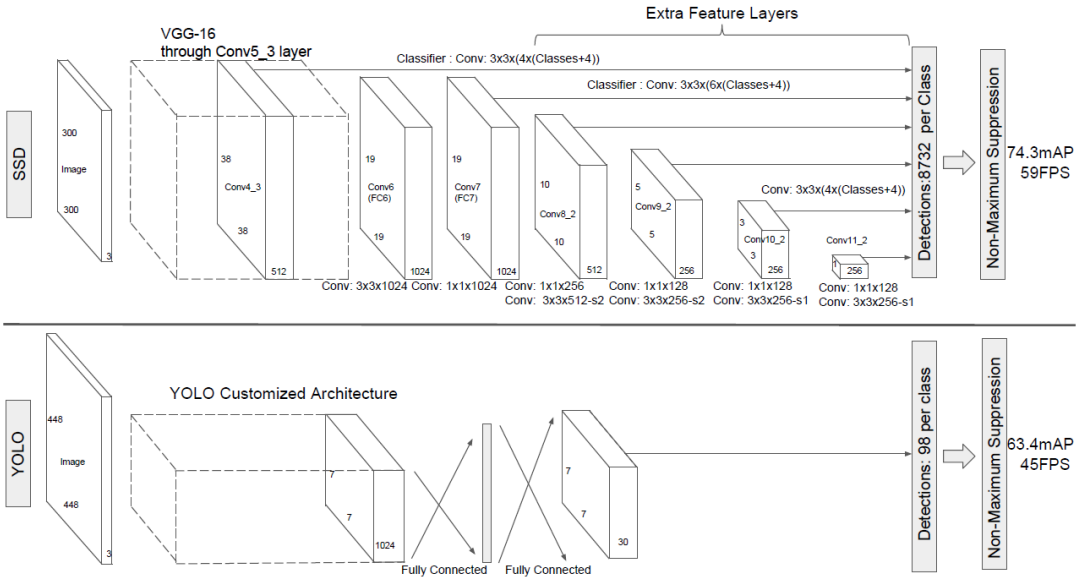


Figura 36: Comparação da arquitetura da rede SSD com YOLO. Imagem adaptada de Liu et al. [2015].
No método de OD SSD o autor Liu et al. [2015] adicionou várias *feature layers* no final da rede base. O SSD com um tamanho de entrada 300x300 supera significativamente a precisão do YOLO equivalente 448x448 no teste VOC2007, além de melhorar consideravelmente a velocidade.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
Fast YOLO	52.7	155	1	98	448×448
YOLO (VGG16)	66.4	21	1	98	448×448
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

Figura 37: Comparação do desempenho entre redes de deteção de objetos. Imagem adaptada de Liu et al. [2015].
O SSD faz mais previsões, e apresenta uma precisão semelhante em comparação com certos métodos de OD para imagens de resolução mais baixa.

- O SSD apresenta um pior desempenho comparado com o Faster R-CNN na deteção de objetos mais pequenos. No SSD, pequenos objetos só conseguem ser detectados em camadas de resolução superior (camadas mais à esquerda). Mas essas camadas contêm recursos de baixo nível, como bordas ou manchas de cor, que contém menos informação para a classificação.
- A precisão do SSD aumenta com o aumento do número de *bounding boxes* mas, em contra partida a velocidade fica comprometida.
- Os *feature maps* em várias escalas melhoram a detecção de objetos em diferentes escalas.
- Melhores *default bounding boxes* ajudam na precisão.

- O SSD possui um erro de localização mais baixo comparado ao R-CNN, mas apresenta mais erros de classificação perante categorias semelhantes.
- O SSD512 tem melhor precisão (2.5%) que o SSD300, mas é executado a 22 FPS em vez de 59 FPS.

O SSD é um *single-shot detector*. Este não possui uma *Region Proposal Network* e prevê as *boundary boxes* e as classes diretamente dos *feature maps* numa única passagem. Para melhorar a precisão, o SSD apresenta:

- pequenos filtros convolucionais para prever classes de objetos e deslocamentos para *boundary boxes* padrão.
- filtros separados para caixas padrão para lidar com a diferença de proporções.
- *feature maps* em várias escalas para detecção de objetos.

Em suma, o SSD pode ser treinado de ponta a ponta para obtenção de uma melhor precisão. O SSD faz boas previsões e tem uma boa cobertura de localização, escala e proporções. Com as melhorias acima, o SSD pode reduzir a resolução da imagem de entrada para 300x300 com um desempenho bom na precisão. Ao remover a *Region Proposal* e ao usar imagens de baixa resolução, o modelo pode ser executado com uma velocidade em tempo real e ainda supera a precisão do Faster R-CNN de última geração.

2.5.5 *You Only Look Once V2 - 2017*

O YOLOv2 foi lançado a janeiro de 2017 por Redmon and Farhadi Redmon and Farhadi [2017], este método é um sistema de OD em tempo real. Este sistema é uma segunda versão do YOLO, sendo que a primeira foi apresentada pelo mesmo autor em 2015.
YOLOv2 ou também chamado de YOLO9000 é um sistema de OD em tempo real que consegue detetar mais de 9000 categorias de objetos. Segundo Redmon and Farhadi Redmon and Farhadi [2017], este método promete maior velocidade e precisão, ultrapassando assim métodos avançados. Esta proposta junta treino e classificação na deteção de objetos, e foi treinada nos *dataset*, COCO e ImageNet simultaneamente. O propósito geral dos OD é que a deteção seja rápida, precisa e capaz de reconhecer uma grande variedade de objetos. Os requisitos mencionados anteriormente só foram conseguidos em simultâneo desde introdução das redes neuronais.
O YOLOv2 é uma abordagem inovadora de OD que supera outros métodos populares como R-CNN. Este método utiliza *labeling images* para aprender a localizar objetos com precisão enquanto usa a classificação para aumentar a sua robustez.
A primeira versão do YOLO sofria de uma variedade de deficiências em relação aos sistemas de detecção mais modernos. A análise de erros do YOLO em comparação com o Fast R-CNN mostra que o YOLO errava várias vezes a localização. Além disso, o YOLO apresentava um *recall* relativamente baixo em comparação com os métodos baseados em propostas de ROI. Assim sendo, o autor nesta segunda versão, YOLOv2, propôs melhorar a recolha e localização de objetos, mantendo a precisão da classificação.
A versão original YOLO recorre a uma estratégia inovadora, o uso da regressão para resolver o problema de OD. Este método deteta as coordenadas das *bounding boxes* e as probabilidades de a classe estar na

imagem. Seguidamente, divide a imagem de entrada com uma grade SxS, onde cada célula da grade prevê B *bounding boxes* e a pontuação de confiança de a classe estar contida nessa célula B, esta pontuação reflete a probabilidade da caixa prevista B conter um objeto de interesse para classificação.
Cada caixa de previsão B é composta por 5 componentes [x,y,w,h,confiança]: (x,y) indicam o centro da caixa em relação à célula da grade correspondente e (w,h) dizem respeito ao peso e altura em relação à imagem inteira.
Se o centro de um objeto cair numa célula de grade, essa célula é responsável por detetar o objeto, de modo que o objetivo de cada célula é detetar o centro do objeto, prevendo assim, um conjunto de probabilidades de classe para cada célula de grade.
O tensor previsto para cada imagem tem a dimensão de é de $S \times S \times (B \times 5 + C)$. Na figura 38, a imagem é dividida por uma grade de 7x7 e prevê B caixas por célula, este modelo é avaliado no *dataset* VOC Pascal em que existem 20 classes, portanto o valor de C=20. Em suma, o tensor de saída é $7x7x(2x5 + C)$.

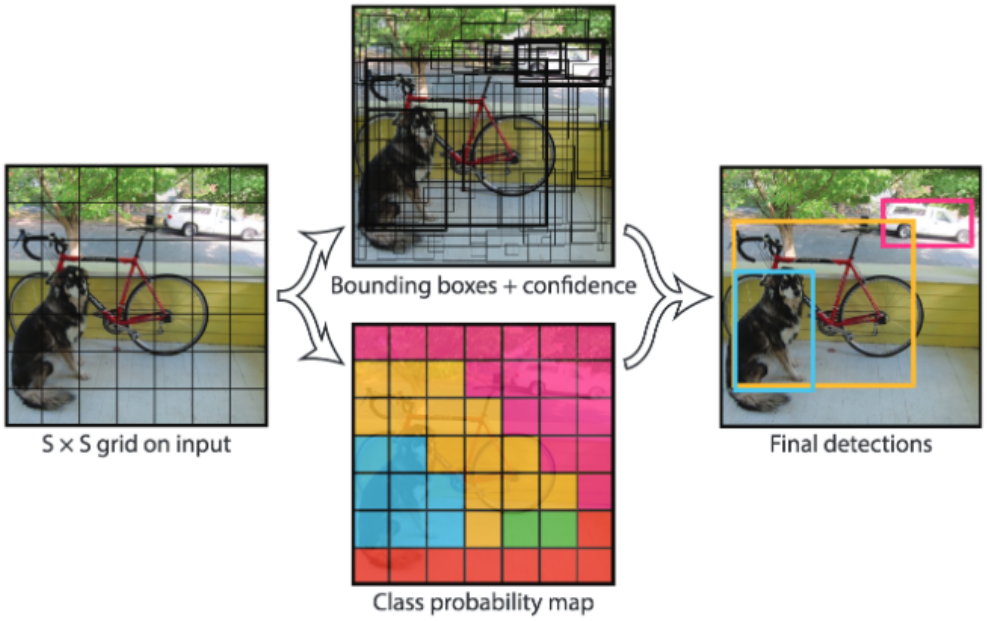


Figura 38: Processo de classificação do Yolov2, predição de *bounding boxes*. Imagem adaptada de Redmon and Farhadi [2017].
A imagem de entrada é divida numa grade de $S \times S$, onde são preditas B *bounding boxes* (regressão). Uma classe é predita baseando-se na qual obteve uma maior confiança entre todas as classes C (classificação).

O modelo atribui a cada caixa uma pontuação de confiança. Essas pontuações avaliam a probabilidade da classe aparecer na caixa e a precisão das coordenadas da respectiva caixa. A primeira versão de YOLO apresentava algumas limitações:

- Mau desempenho na deteção de objetos pequenos;
- Como cada célula da grelha prevê apenas duas classes e pode ter apenas uma, isso limita o número de objetos próximos que o YOLO consegue prever, especialmente para objetos pequenos que aparecem em grupos, como por exemplo um bando de pássaros.

- Como é usada uma grelha 7x7, e qualquer grade pode detectar apenas um objeto, o número máximo de objetos que o modelo consegue detectar é 49.

- Erro de localização relativamente alto.

Devido às limitações mencionadas anteriormente, foi então desenvolvido o YOLOv2 que apresenta um sistema de detecção de objetos em tempo real que detecta mais de 9000 categorias de objetos, corrigindo ainda os erros de localização e os valores baixos de *recall*. Esta segunda versão utiliza a *Batch Normalization* que leva a uma significativa melhoria na convergência, eliminando a necessidade de outras formas de regularização. Ao adicionar a *Batch Normalization* em todas as camadas convolutivas foram obtidos mais de 2% de melhoria no mAP.
O YOLOv2 pré-treina o classificador no ImageNet usando entradas de resolução 224x224, de seguida, aumenta a resolução para 448x448 para a detecção. Este processo requer tempo de rede para se ajustar simultaneamente à nova entrada de resolução e executar a tarefa de detecção.
No entanto, o YOLOv2 resolve isso da seguinte maneira:

- Primeiro o classificador é treinado na resolução 224x224 no ImageNet.

- Em segundo lugar, o classificador é treinado na resolução de 448x448 por 10 *epochs*, fazendo com que os filtros da rede se ajustem às entradas de maior resolução. Em seguida, a rede resultante é ajustada na detecção. Este ajuste do classificador em entradas de alta resolução aumenta a precisão em 4% mAP.

Esta versão recorre também ao uso de *anchor boxes* que são um conjunto de caixas com formas predefinidas. Estas caixas tem uma altura e uma proporção de largura típicas. Portanto, ao prever *bounding boxes* apenas se ajusta o tamanho dessas *anchor boxes* para que se encaixem nos objetos. O uso de *anchor boxes* facilita muito o processo de aprendizagem do modelo.
Com o uso de *anchor boxes*, o YOLOv2 melhorou o *recall* em 7%, o que significa que aumentou a percentagem de casos positivos, mas diminuiu a precisão em uma pequena margem.
Outra tecnologia usada são os *clusters* de dimensão. Estes seleccionam as dimensões das *anchor boxes*, executando o algoritmo K-means no conjunto de treino para encontrar os melhores valores para as principais *anchor boxes*. Para encontrar a *anchor boxes* mais próxima, procura-se que esta tenha boa pontuação de IoU com a *bounding box*.
O objetivo deste algoritmo é encontrar diferentes valores de *cluster* K, ver Figura 39. O autor percebeu que K=5 é o melhor valor, pois oferece uma boa relação entre a complexidade do modelo e um *recall* relativamente alto (ou seja, percentagem de casos positivos).
Para a previsão da localização do objeto o YOLOv2 prevê 5 parâmetros (tx, ty, tw, th, eto) onde é aplicada a função *sigma* para calcular o deslocamento possível entre a *anchor box* e a *bounding box*. Na Figura 40 pode-se observar uma caixa azul que representa a *bounding box* prevista e o rectângulo pontilhado que representa a *anchor box*. Com o uso do *cluster* de *k-means* (clusters de dimensão), o mAP aumenta 5%.

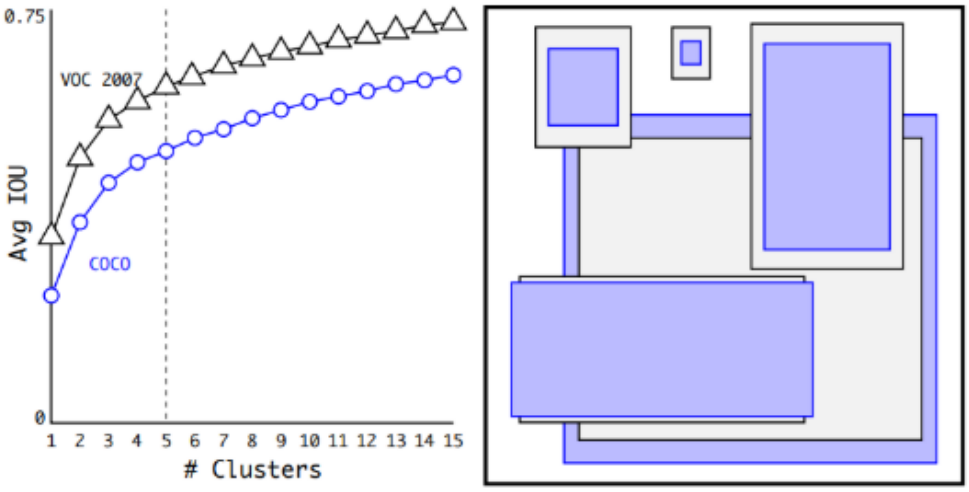


Figura 39: Cálculo do número de *anchor boxes* para o *dataset* COCO e Pascal VOC. Imagem adaptada de Redmon and Farhadi [2017].
Para diferentes números de *clusters* k é observado no gráfico o valor de IoU, o autor conclui que $k = 5$ oferece uma boa compensação analisando a complexidade do modelo versus o *recall*. No gráfico à direita estão apresentadas as 5 formas de *anchor boxes* seleccionadas para o *dataset* COCO a azul e Pascal VOC a cinza.

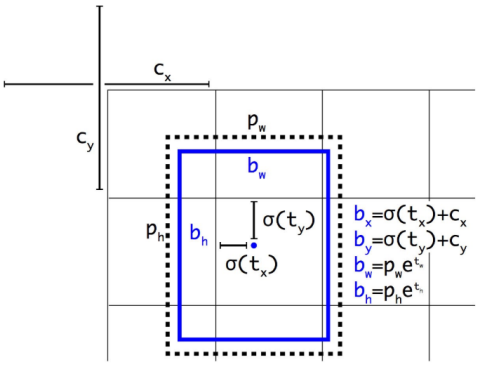


Figura 40: Esquema da previsão de localização das *bounding box* com uso das *anchor box*. Imagem adaptada de Redmon and Farhadi [2017].

2.5.6 *You Only Look Once V3 - 2018*

YOLOv3 consiste na versão aprimorada do YOLOv2, esta nova versão foi apresentada em 2018 por Redmon and Farhadi Redmon and Farhadi [2018], e apresenta um desempenho consideravelmente superior.
As melhorias implementadas consistem no uso de um melhor extrator de *features*, Darknet-53 e num melhor detector de objetos.
Segundo Kathuria Kathuria [2019], o YOLOv2 usou uma arquitetura profunda, o Darknet-19, que é uma rede original de 19 camadas complementada com mais 11 camadas para a detecção de objetos. Com uma arquitetura de 30 camadas, o YOLOv2 enfrentava frequentemente a necessidade de classificar objetos de pequenas dimensões, apresentando dificuldade nesta tarefa, o autor Redmon and Farhadi Redmon and Farhadi [2018]

incorpora a resolução deste problema no YOLOv3.

Primeiro, o YOLOv3 usa uma variante do Darknet, que originalmente possui uma rede de 53 camadas treinada no Imagenet. Para a tarefa de detecção, são empilhadas mais 53 camadas, fornecendo uma arquitetura subjacente totalmente convolucional de 106 camadas.

Na Figura 41 é apresentada uma proposta de apresentação da arquitetura do YOLOv3, Kathuria [2019].

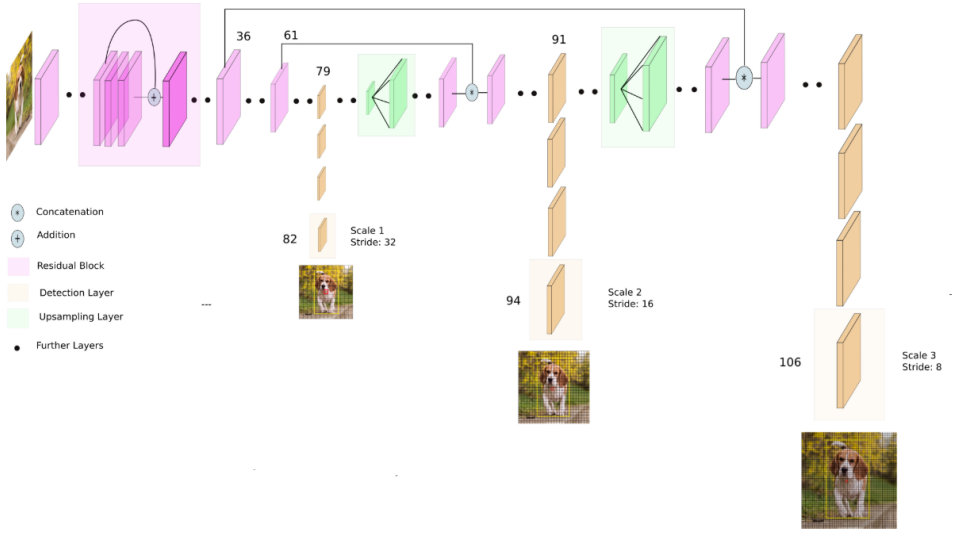


Figura 41: Representação da arquitetura do método de OD YOLOv3. Imagem adaptada de Kathuria [2019].

YOLOv3 é invariável para o tamanho da imagem de entrada. Um problema apresentado é o de processar imagens em *batch* (as imagens em *batch* podem ser processadas em paralelo pela GPU, levando ao aumento da velocidade), para isto, é necessário que as imagens apresentem altura e largura fixas. Isso é necessário para concatenar várias imagens numa *batch*.

A rede reduz a amostragem da imagem por um fator chamado passo da rede. Por exemplo, se o passo da rede for 32, uma imagem de entrada do tamanho 416x416 produzirá uma saída do tamanho 13x13. Geralmente, o passo de qualquer camada da rede é igual ao fator pelo qual a saída de a camada é menor que a imagem de entrada na rede.

A técnica usada na previsão de *Bounding Boxes* no YOLOv3 é a mesma utilizada pelo YOLOv2. Mas na previsão de Classes, o *softmax* não é utilizado como no YOLOv2.

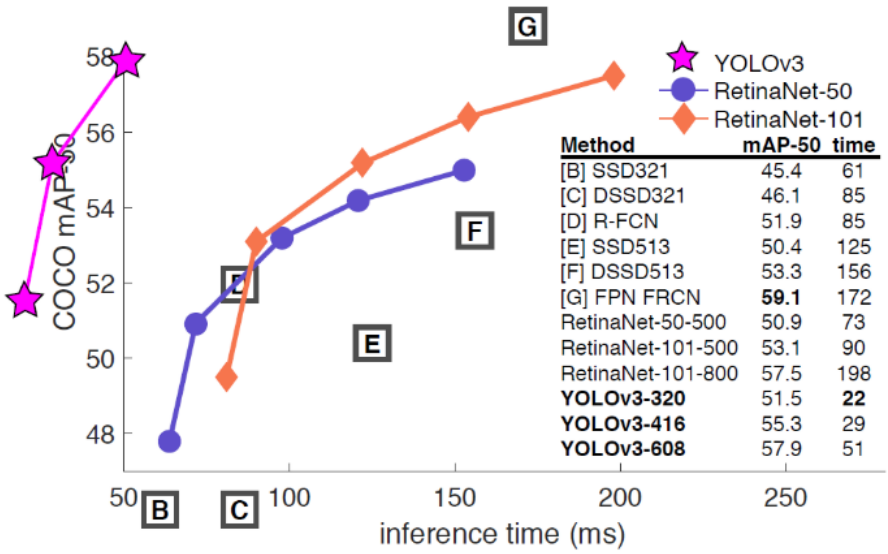
O YOLOv3 faz previsões em três escalas, que são dadas precisamente ao reduzir a amostragem das dimensões da imagem de entrada em 32, 16 e 8, respectivamente, ver Figura 41. No YOLOv3, são usadas no total 9 *Anchor Boxes*. Três para cada escala. Detecções em diferentes camadas ajudam no problema de deteção de pequenos objetos.

Na Figura 42 o autor Redmon and Farhadi Redmon and Farhadi [2018] apresenta um comparativo de *backbones*. Comparando o Darknet-53 com ResNet-101, o *backbone* usado no YOLOv3 apresenta um melhor desempenho, o autor refere que este é 1.5 vezes mais rápido. Comparado com o ResNet-152, o Darknet-53 tem um desempenho semelhante mas é 2x mais rápido, Redmon and Farhadi [2018].

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Figura 42: Gráfico comparativo de *Backbones*. Imagem adaptada de Redmon and Farhadi [2018].

Na figura 43, é comparado o método o YOLOv3 com o RetinaNet, o YOLOv3 obteve um mAP comparável mas, com um tempo de inferência muito mais rápido. Por exemplo, o YOLOv3-608 obteve 57.9% de mAP em 51ms, enquanto o RetinaNet-101–800 obteve apenas 57,5% de mAP em 198ms, o que o torna 3.8 vezes mais rápido.



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figura 43: Gráfico interpretativo do desempenho do método YOLOv3 em comparação com outro métodos populares. Comparação de *mean Average Precision* e tempo entre métodos. Imagem adaptada de Redmon and Farhadi [2018].

Em suma, o YOLOv3 é muito melhor que o SSD e apresenta um desempenho semelhante ao DSSD. O autor Tsang Tsang [2019b], refere que não existem resultados qualitativos suficientes para dar preferência a um só método, pelo que recomenda alternar entre o YOLOv2 e YOLOv3 consoante o problema que se pretende resolver, e assim analisar e escolher qual o método mais eficiente.

2.6 Semantic segmentation

A Semantic Segmentation é o processo de atribuição de uma classe a cada pixel constituinte de uma imagem. Existem dois tipos de Segmentation: a Semantic Segmentation e a Instance Segmentation.
Na Semantic Segmentation todos os objetos da mesma classe numa imagem são atribuídos com a mesma classe, já na Instance Segmentation os objetos diferentes da mesma classe são rotulados com classes dife-

rentes, para uma melhor compreensão observar a Figura 44.

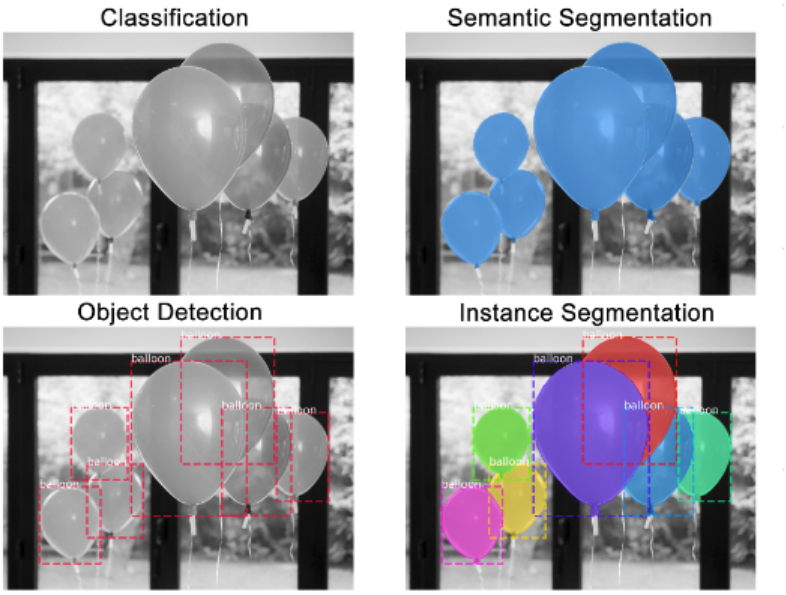


Figura 44: Esquema explicativo da diferença entre Semantic Segmentation e Instance Segmentation. Imagem adaptada de Dwivedi [2019].
Para compreensão da diferença entre os métodos anteriormente referidos, é preciso observar a imagem original que se pretende classificar (que contém balões em tons de cinza). Na Semantic Segmentation todos os balões da imagem original são classificados com a mesma classe (a azul), já na Instance Segmentation cada balão da imagem original obteve uma classe diferente, todas estas classes (de diferentes cores) se referem a diferentes balões. Com a Instance Segmentation é possível diferenciar os balões dentro da imagem original. Na figura, é ainda possível observar como é feita a classificação através de métodos de OD mencionados na secção 2.5.

Em seguida, serão apresentados alguns estudos de modelos de Semantic Segmentation presentes no Estado de arte.

2.6.1 *U-Net - 2015*

A U-Net é uma arquitetura de Redes Convolucionais desenvolvida em 2015 por Ronneberger et al. [2015], para segmentação de Imagens Biomédicas.
Segundo Lamba [2019], esta arquitetura é composta por duas partes (dois caminhos), ver Figura 45. O primeiro caminho é o caminho de contração (também chamado de Codificador) que é usado para capturar o contexto da imagem. O Codificador é uma pilha tradicional de camadas convolucionais e de pool máximo. O segundo caminho é o caminho de expansão simétrico (também chamado de Descodificador), que é usado para permitir a localização precisa, usando para isso convoluções transpostas. Por fim, os diferentes andares de codificação e descodificação são conectados com ligações residuais. Portanto, a U-Net trata-se de uma *Fully Convolution Network (FCN)*, ou seja, contém apenas camadas convolucionais, pelo que pode aceitar imagens de qualquer tamanho.
Na figura 45 pode-se observar que o tamanho da imagem de entrada é 572x572x3, no entanto é usado o

tamanho de entrada 128x128x3. Com isto, conclui-se que o tamanho da imagem varia entre vários locais na rede da U-Net mas os componentes principais permanecem os mesmos, Lamba [2019].

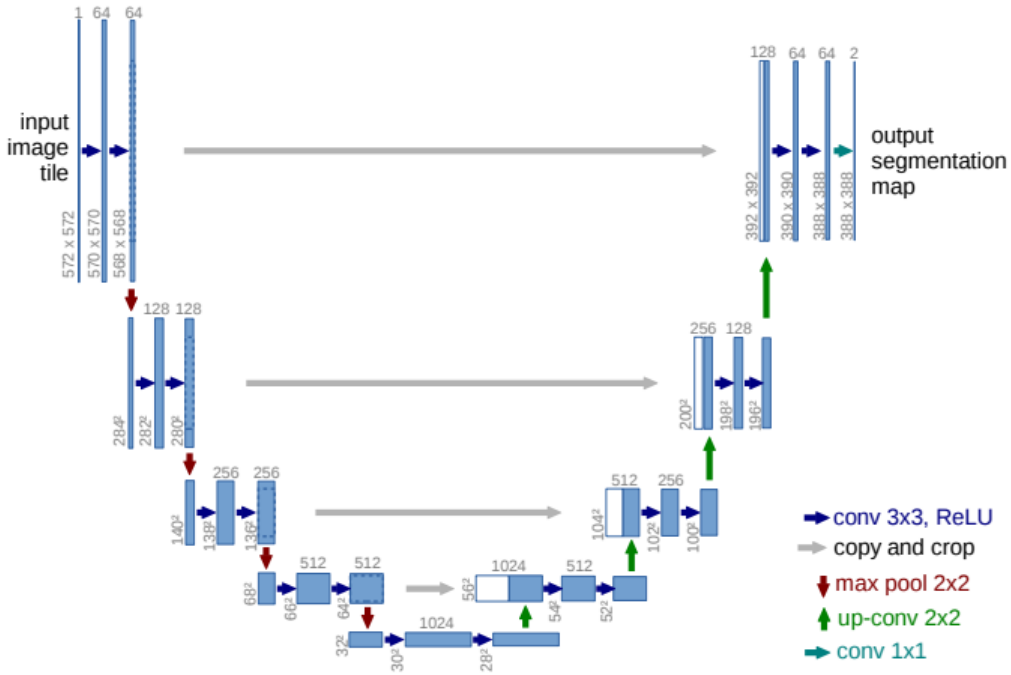


Figura 45: Esquema representativo da arquitetura U-Net (exemplo para 32x32 pixeis na resolução mais baixa). Imagem adaptada de Ronneberger et al. [2015].
Cada caixa azul corresponde a um *feature map* de multi-canais. O número de canais é indicado em cima da caixa. O tamanho x-y é fornecido na borda inferior esquerda da caixa. Caixas brancas representam *feature maps* copiados. As setas indicam as diferentes operações.

A U-Net recorre também a *Data augmentation*, que é muito importante na tarefa de ensinar à rede as propriedades tão desejadas de invariância e robustez. O uso da *data augmentation* é importante quando se está perante um pequeno leque de amostras disponíveis para treino.
O autor Ronneberger et al. [2015] conclui que, a arquitetura U-Net atinge um desempenho muito bom em aplicações bastante diferentes de segmentação biomédica.

2.6.2 *DeepLabv3+ - 2018*

No documento referenciado por Chen et al. [2018] lançado em 2018, é proposto um método de Semantic Segmentation, que usa o modelo DeepLabV3+ da Google. A arquitetura do método DeepLabV3+ é composta por duas fases: numa primeira fase ocorre uma codificação e numa segunda fase uma descodificação, ver Figura 46.

- Codificador: Nesta etapa, uma CNN pré-treinada extrai as informações essenciais da imagem de entrada. Para tarefas de segmentação, as informações essenciais são os objetos presentes na imagem e a sua localização.

- Descodificador: As informações extraídas da fase de codificação são usadas para criar uma saída com o tamanho original da imagem de entrada.

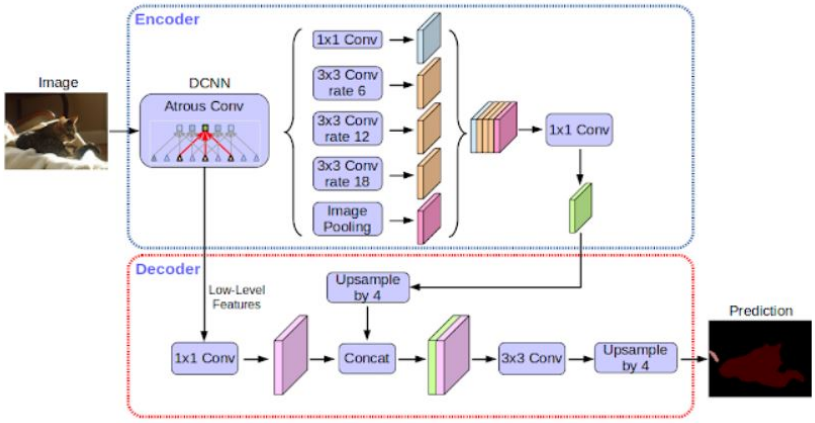


Figura 46: Arquitetura da DeepLabV3+. Imagem adaptada de Chen et al. [2018].
O modelo DeepLabV3+ proposto pelo Chen et al. [2018] é uma extensão do antigo DeepLabV3, para formar a nova versão do método o autor acrescentou uma estrutura codificador-descodificador. O módulo codificador codifica informações contextuais em várias escalas aplicando *atrous convolution* em várias escalas também, enquanto o módulo descodificador serve para refinar os resultados da segmentação ao longo dos limites do objecto.

O autor Chen et al. [2018], no desenvolvimento do método DeepLabV3+, recorreu a dois tipos de redes neuronais que usam *spatial pyramid pooling module* e estrutura *encoder-decoder* para Semantic Segmentation, em que o primeiro captura boas informações contextuais ao agrupar as *features* em diferentes resoluções e o segundo é capaz de obter limites nítidos de objetos.
Para capturar informações contextuais em várias escalas, o DeeplabV3 aplica várias *parallel atrous convolution* com taxas diferentes ASPP. O DeepLabV3+, é estendido com a adição de um módulo descodificador simples, porém eficaz, para recuperar os limites do objeto, como esta ilustrado na Figura 47. As principais

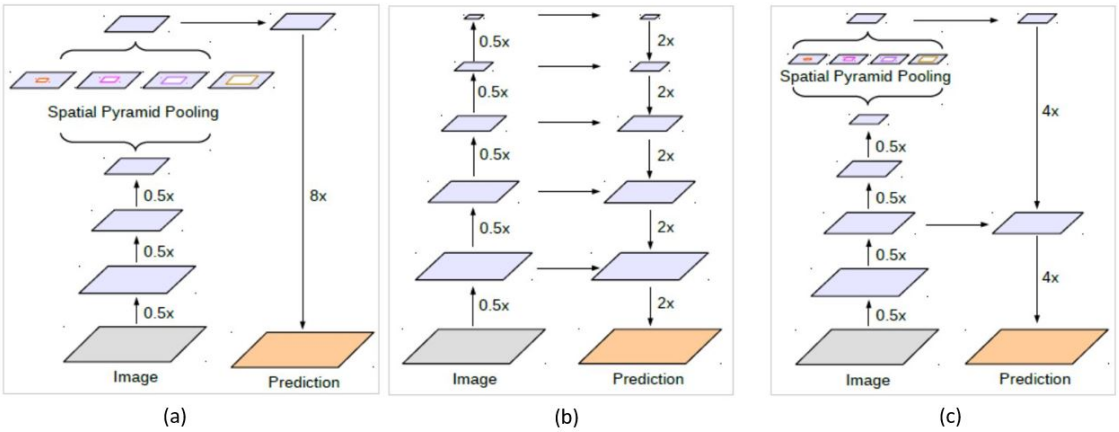


Figura 47: (a) Atrous Spatial Pyramid Pooling ASPP consegue codificar informações contextuais em várias escalas, (b) Arquitetura codificador-decodificador, as informações de localização/espaço são recuperadas, esta arquitetura provou ser útil em literatura como FPN , DSSD , TDM , SharpMask , RED-Net e U-Net para diferentes tipos de finalidades, (c) O DeepLabv3+ faz uso de (a) e (b). Imagem adaptada de Tsang [2019a].

contribuições propostas pelo autor são:

- Uma nova estrutura de codificador-descodificador que utiliza o DeepLabV3 como um poderoso módulo de codificador e ainda um módulo de descodificador simples e eficaz.
- Na estrutura proposta é possível controlar arbitrariamente a resolução das *features* do codificador extraído por *atrous convolution* para compensar a precisão e o tempo de execução, o que não é possível nos modelos existentes de codificador-descodificador.
- O modelo Xception foi adaptado para a tarefa de segmentação e foi aplicado *depthwise separable convolution* ao módulo ASPP e ao módulo do descodificador, resultando assim, uma rede de codificador-descodificador mais rápida e forte.
- O modelo proposto obtém um novo desempenho com alta performance nos *datasets* PASCAL VOC 2012 e Cityscapes.

Em suma, o modelo proposto DeepLabV3+ emprega uma estrutura codificador-descodificador no DeepLabV3. O codificador serve para codificar as informações contextuais e o descodificador que é um processo simples, serve para fazer a recuperação dos limites dos objetos. No método pode também aplicar-se a *atrous convolution* para extrair as *features* do codificador numa resolução à escolha, dependendo dos recursos de computação disponíveis. O autor explora ainda o modelo Xception e a *atrous separable convolution* para tornar o modelo proposto mais rápido e forte. Finalmente, os resultados experimentais mostram que o modelo proposto apresenta um alto desempenho nos *datasets* PASCAL VOC 2012 (ver Figura 48) e Cityscapes.

Method	mIOU
Deep Layer Cascade (LC) [82]	82.7
TuSimple [77]	83.1
Large_Kernel_Matters [60]	83.6
Multipath-RefineNet [58]	84.2
ResNet-38_MS_COCO [83]	84.9
PSPNet [24]	85.4
IDW-CNN [84]	86.3
CASIA_IVA_SDN [63]	86.6
DIS [85]	86.8
DeepLabv3 [23]	85.7
DeepLabv3-JFT [23]	86.9
DeepLabv3+ (Xception)	87.8
DeepLabv3+ (Xception-JFT)	89.0

Figura 48: Comparação de abordagens entre o Deeplabv3+ e outros métodos poderosos. Todos os métodos apresentados foram testados com o *dataset* PASCAL VOC 2012. Imagem adaptada de Tsang [2019a].

3

ESTUDO E SELEÇÃO DOS SENSORES PARA CARACTERIZAÇÃO DOS DANOS

Este capítulo é dedicado à explicação detalhada de todo o processo efetuado na criação dos *datsets out-car* e *in-car*. O *dataset out-car* serve como prova de conceito, ou seja, serão avaliados métodos de classificação de danos primeiramente neste *dataset* e consoante o resultado deste estudo, os melhores serão avaliados numa fase final no *dataset in-car*. Uma versão preliminar da criação do dataset in-car é apresentada nos artigos "**In-Car Damage and Stain Estimation with RGB Images**"e "**In-Car State Classification with RGB Images**".
Inicialmente é feita a apresentação dos sensores utilizados transversalmente na geração dos *datasets*.
Seguidamente são apresentados, individualmente, os *datasets out-car* e *in-car*. Onde em cada um deles é feita a caracterização de danos e materiais existentes no interior do veiculo. Concluindo com os procedimentos e ferramentas desenvolvidos para a geração e gestão destes *datasets*.

3.1 Sensores utilizados para a deteção de danos

Na secção 2.3 foram apresentadas metodologias TND baseadas em sensores RGB e IR. Muitas das metodologias do estado da arte fazem uso de sensores RGB, alternativamente a técnica Lock-In termográfica apresenta novas oportunidades de inspeção. Contudo, esta metodologia apresenta uma elevada complexidade na extração de features (i.e. excitação termica com modulação, captura sincronizada de imagens termicas, e calculo de imagens Lock-In), e por essa razão o foco deste trabalho será depositado na utilização de sensores RGB.
Para a realização do presente trabalho é necessário responder a duas questões pertinentes:

- Qual é o sistema mínimo de inspecção num cenário automóvel?
- Qual é a melhor abordagem algorítmica para garantir a deteção de danos, manchas e sujidade com o sistema mínimo de inspecção?

Para responder a estas perguntas, o ponto fulcral incide na escolha e aquisição de bons sensores RGB e iluminação especifica para um comprimento de onda dentro do cenário automóvel.
Como foi demonstrado na secção 2.3.2, quase todas as metodologias de inspecção seguem um registo muito semelhante, com uma fonte de iluminação/excitação e um sensor de captura de imagem.
Para fornecer um sistema capaz de detectar danos, sujidade e manchas dentro de um veículo, é necessário fornecer um sistema de teste com sensores/iluminação que sejam capaz de capturar esses objetos. Esse sistema também exigirá uma visualização ampla do cenário no interior do carro, que num primeiro estágio

pode ser considerada excessiva, no entanto, proporcionará uma melhor compreensão da melhor configuração do sensor para uma solução minimizada. Os sensores serão usados para capturar *datasets* em posições específicas do carro.

Os sensores RGB são uma escolha mais fácil, devido ao seu baixo custo, maior campo de visão e alta resolução. Para a criação do *dataset out-car* recorreu-se ao sensor OMRON FZ-SC2M. Este sensor é proveniente de um sistema de visão industrial da OMRON FH, sistema este que serve como interface para captura das informações (*features*) provenientes do sensor. A criação do *dataset out-car* é importante no sentido da validação dos algoritmos para deteção de danos no interior do veiculo.

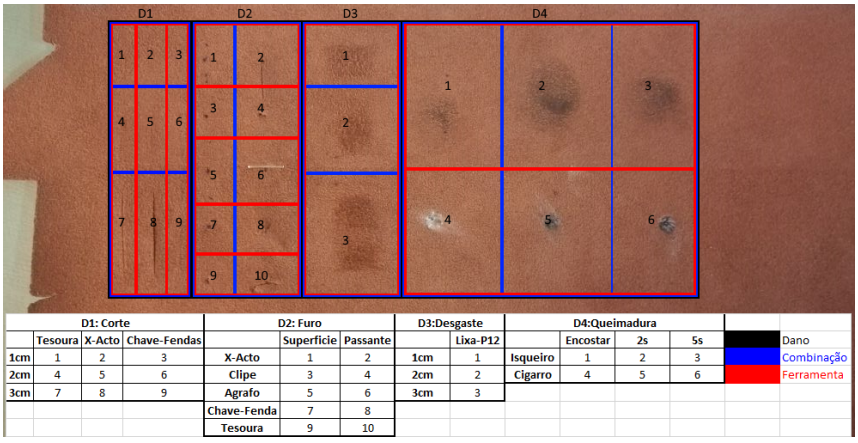
O *dataset in-car* será capturado através de sensores RGB disponíveis. A recolha das imagens será feita em carros disponíveis em sucatas que apresentam facilmente todos os objetos necessários para a inspecção, algumas imagens serão de carros semi-novos capturadas em stands automóveis. Os sensores RGB disponíveis apresentam HDR, alta resolução (maior que 8Mpixeis) e um campo de visão *ultrawide* (maior que 110°).

3.2 Dataset out-car

3.2.1 *Caraterização dos tipos de danos em estudo*

Nesta dissertação, recorre-se ao uso de um calibrador, ver Figura 49, o uso deste calibrador considera-se bastante válido pois este abrange uma amostra de danos bastante satisfatória e alargada que podem ocorrer no interior do veículo. Este calibrador foi criado para gerar danos em materiais existentes no interior dos veículos. Permitindo a criação de um *dataset out-car*.

Numa fase final, o *dataset* irá suportar a primeira etapa de desenvolvimento algorítmico, permitindo assim a seleção dos melhores candidatos de deteção de danos.



D1: Corte			
	Tesoura	X-Acto	Chave-Fendas
1cm	1	2	3
2cm	4	5	6
3cm	7	8	9

D2: Furo		
	Superficie	Passante
X-Acto	1	2
Clipe	3	4
Agrafo	5	6
Chave-Fenda	7	8
Tesoura	9	10

D3:Desgaste	
	Lixa-P12
1cm	1
2cm	2
3cm	3

D4:Queimadura			
	Encostar	2s	5s
Isqueiro	1	2	3
Cigarro	4	5	6

Dano
Combinação
Ferramenta

Figura 49: Template de danos a aplicar as diferentes amostras de materiais.

O calibrador está dividido em quatro zonas: em que (D1) corresponde ao corte, (D2) corresponde ao furo, (D3) corresponde ao desgaste e por último, (D4) corresponde às queimaduras. Este calibrador foi aplicado em seis amostras de tecido. Teve-se em conta os materiais comummente utilizados dentro de um veículo na escolha das amostras em estudo. Estas amostras serão apresentadas na Secção 3.2.2.

Na zona D1, é apresentado o corte, foram aplicados cortes de três naturezas, provenientes de x-ato, tesoura e

chave de fendas. Com estas ferramentas fizeram-se cortes com 1 centímetro, 2 centímetro e 3 centímetros.
Na zona D2, é apresentada a perfuração, foram aplicadas cinco tipos de ferramentas diferentes: x-ato, chave de fendas, clipe, agrafo e tesoura. Existem dois tipos de perfuração, a superficial e a passante, na superficial a ferramenta é passada sobre o tecido, na passante o objetivo é que a ferramenta atrevesse todas as camadas de tecido.
Na zona D3, é apresentado o desgaste, para simular este dano muito comum num carro passados alguns anos foi utilizada uma lixa. Foram utilizadas lixas com diferentes comprimentos, [1;2;3] centímetros. Com o aumento do comprimento da lixa mais profundo é o desgaste provocado.
Na zona D4, é apresentada a queimadura, para provocar este dano utilizaram-se dois instrumentos, um cigarro e um isqueiro. Nesta secção e com estes dois instrumentos fizeram-se 3 ensaios. No primeiro, encostaram-se ambas as ferramentas ao tecido e retiraram-se imediatamente. No segundo, aguardou-se 2 segundos. Em terceiro e por ultimo aguardou-se 5 segundos. Quanto maior o tempo de contacto da ferramenta quente com a pele, maior e mais visível é o dano provocado.

3.2.2 *Materiais utilizados*

Com o intuito de inspeccionar os danos em diferentes tipos de materiais existentes no interior do veículo, foram criadas 10 amostras, $S_{1:10}$, (ver Figura 50) com diferentes tipos de material:

- S_1 - Anilina bobina;
- S_2 - Por identificar;
- S_3 - Laminado: malha revestida acabamento carbono (Esmerizazo);
- S_4 - Por identificar;
- S_5 - Malha revestida coating;
- S_6 - Laminado: malha, espuma e malha;
- S_7 - Anilina bobina;
- S_8 - Polímero PVC;
- S_9 - Revestimento malha polímero coating;
- S_{10} - Laminado: malha revestida acabamento carbono (Esmerilada).

3.2.3 *Geração de dataset out-car*

Para a geração do *dataset out-car* foram utilizadas 8 das 10 amostras apresentadas na secção 3.2.2, as amostras utilizadas foram $[S_1, S_2, S_3, S_4, S_7, S_8, S_9, S_{10}]$. As imagens foram recolhidas a partir do sensor OMRON FZ-SC2M, a resolução das imagens recolhidas é de 1200x1600, o *setup* de recolha de imagens usado para criação do *dataset* pode ser visto na Figura 51.

De modo a diversificar as imagens recolhidas colocou-se a câmara a partir de 4 alturas diferentes em relação à amostra e cada amostra foi captada em várias perspectivas e diferentes rotações. As alturas escolhidas foram [16.5; 21; 25; 29]cm, para cada altura foram recolhidas 50 imagens por amostra, em diferentes posições, o que perfaz um total de 200 imagens por amostra. Contabilizando 8 amostras, o número total de imagens capturadas é 1600.

O passo seguinte para a criação do *dataset* é a atribuição de *labels* às classes de interesse que estão nas imagens. Para efectuar esta tarefa recorreu-se à aplicação *Ground Truth Labeler* disponível no MATLAB versão R2019b. A aplicação permite fazer *labeling* de dados em sequências de imagens ou em vídeo. O *labeling* pode ser feito de três formas, ou seja, as classes podem ser discriminadas em uma imagem de três formas distintas: retangular ROI, linha poligonal ROI e atribuição da classe ao nível do pixel. Ver exemplos na Figura 52.

Na criação do *dataset* optou-se por fazer o *labeling* ao nível do pixel, porque esta forma de atribuição de classe é mais versátil. Existem métodos de classificação que precisam do *labeling* atribuído ao nível do pixel como é o caso dos métodos de *Semantic Segmentation* (Figura 53 (a)), e existem métodos que necessitam do *labeling* usando *bounding boxes* como é o caso dos métodos de *Object Detection* (Figura 53 (b)). Ao escolher fazer a atribuição da classe ao nível do pixel, consegue-se posteriormente avaliar facilmente o *dataset* tanto em métodos de *Semantic Segmentation* como em métodos de *Object Detection*, bastando para isso, desenvolver um algoritmo que encontre o *labeling* ao nível do pixel de uma determinada classe e calcule uma *bounding box* que corresponda a essa classe. O contrário já não seria possível com a mesma facilidade e caso fosse possível prejudicaria muito o desempenho dos métodos de *Semantic Segmentation*, porque estaria a atribuir a classe ao nível do pixel à *bounding box* completa.

Para uma melhor interpretação ver Figura 52, onde é possível ver que a passagem do tipo de *labeling* de (c) para (a) é eficiente, já o contrário não (de (a) para (c)).

Na aplicação *Ground Truth Labeler* criaram-se 4 classes de *pixel label* para as ROIs, ver canto superior esquerdo da Figura 54. As classes são as descritas na secção 3.2.1: Corte, Furo, Desgaste e Queimadura. Uma vez criadas as classes e carregadas as imagens para o ambiente *Ground Truth Labeler*, basta seleccionar a classe na aplicação e desenhar na imagem com o auxilio da ferramenta *brush* a área que se quer atribuir à classe, ver Figura 54. Este procedimento é realizado para todas as 1600 imagens captadas anteriormente.

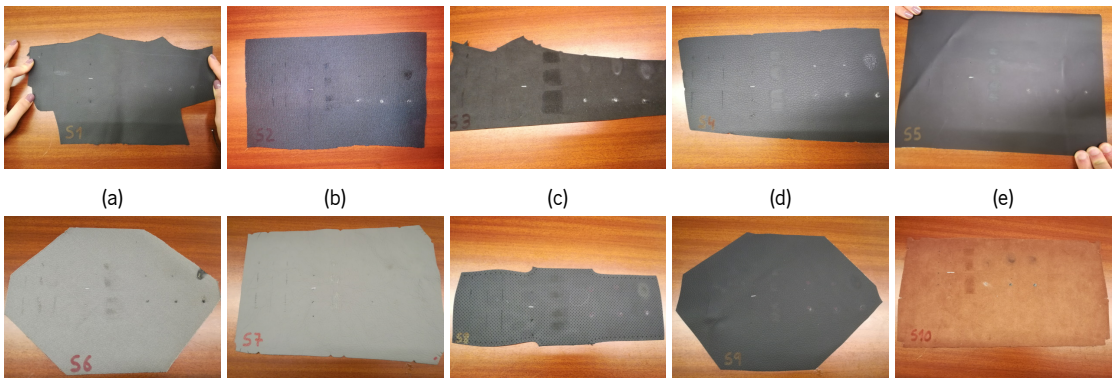


Figura 50: Amostras de tecido tipo pele utilizadas no estudo. (a) S_1, (b) S_2, (c) S_3, (d) S_4, (e) S_5, (f) S_6, (g) S_7, (h) S_8, (i) S_9, (j) S_{10}.

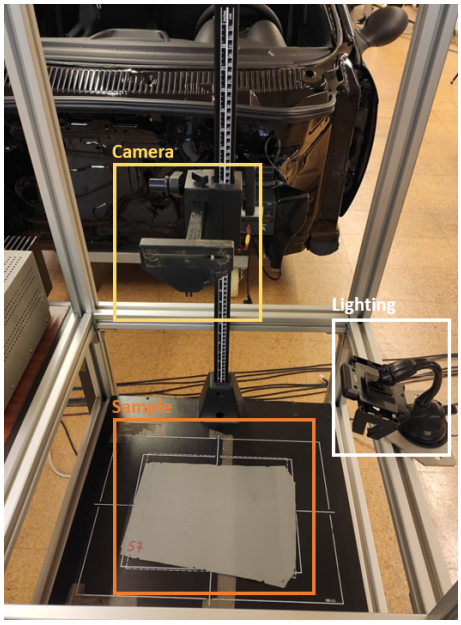
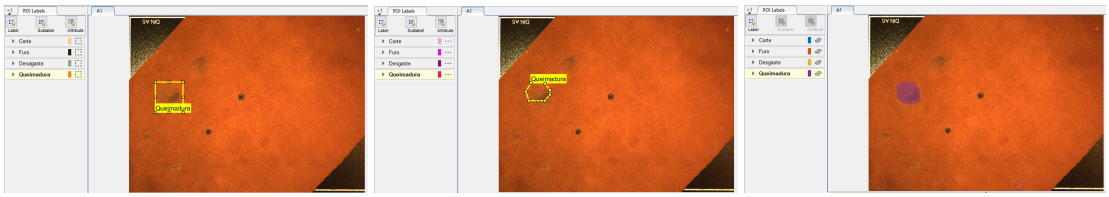
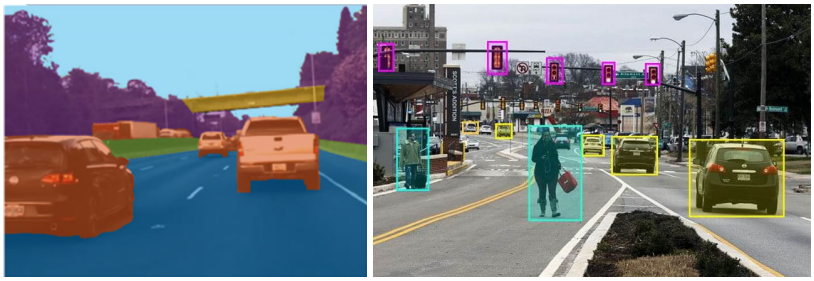


Figura 51: *Setup* de recolha de imagens *out-car*.



(a) (b) (c)

Figura 52: (a) Atribuição da classe à ROI de forma rectangular, (b) Atribuição da classe à ROI através de linha poligonal, (c) Atribuição da classe à ROI ao nível do pixel.



(a) (b)

Figura 53: (a) *Labeling* atribuído à ROI ao nível do pixel, (b) *Labeling* atribuído à ROI de forma rectangular. Imagens adaptadas do site MATLAB.

Finalizada a atribuição das classes para todas as imagens a aplicação cria um ficheiro *.mat* denominado *gTruth*, onde é criada uma tabela com o nome de cada classe e o *PixelLabelID* associado a essa classe, ver Tabela 2.

Na Figura 55, podemos observar na imagem (a) a imagem original da amostra com danos, e na imagem (b) a mascara criada com o processo do *Ground Truth Labeler*, em que é atribuído ao *background* o ID de 0 e

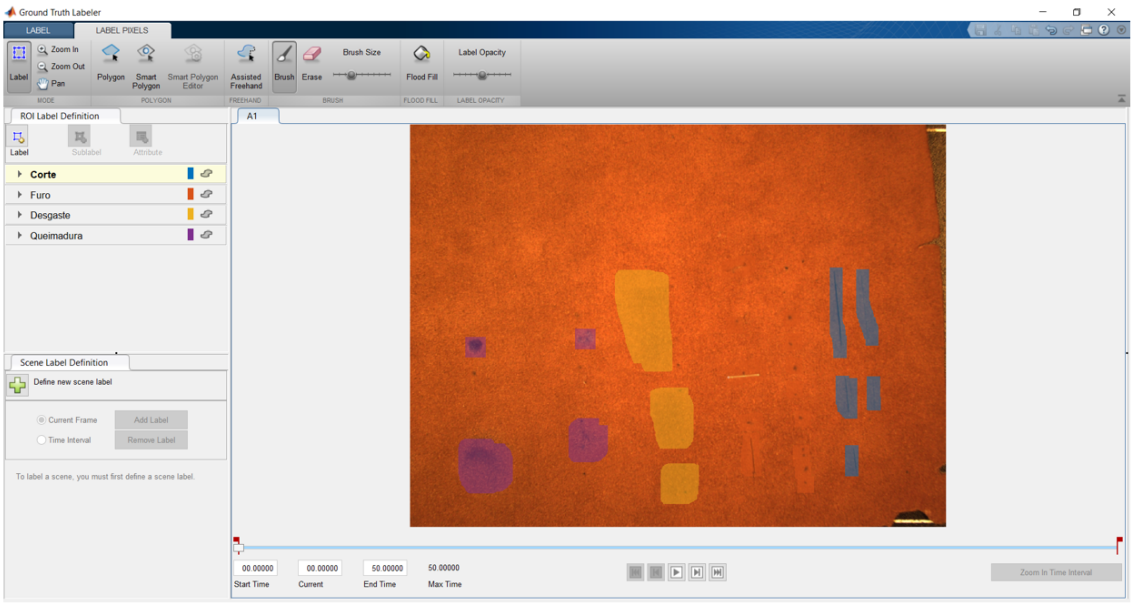


Figura 54: Imagem ilustrativa do funcionamento da aplicação *Ground Truth Labeler* no MATLAB. No lado esquerdo da figura estão apresentadas as classes criadas, em que para cada classe é atribuída uma cor, na parte superior da figura está seleccionada a ferramenta *brush* que foi utilizada para discriminar as classes na imagem da amostra S_{10}. Neste exemplo a imagem da amostra S_{10} apresenta todas as classes e pode-se observar o desenho da classe por cor. A classe queimadura é representada a lilás, desgaste a amarelo, furos a laranja e cortes a azul.

Tabela 2: ID atribuido a cada classe do *dataset out-car*.

Nome Classe	Pixel Label ID
Corte	1
Furo	2
Desgaste	3
Queimadura	4

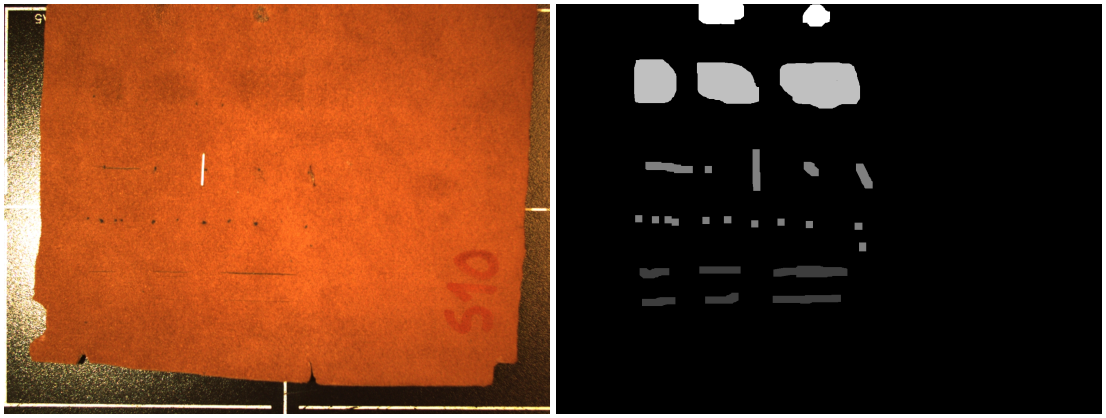
às classes os IDs da Tabela 2. Através de um ajuste para níveis de cinza é possível observar a Figura 55 (b).

3.2.4 *Divisão do dataset para avaliação algorítmica*

Uma vez criado o *dataset out-car* e realizado o *labeling*, surge a necessidade de fazer a divisão do *dataset out-car* para ser usado pelos diferentes algoritmos. Para tal é necessário criar um algoritmo genérico que seja capaz de dividir o *dataset out-car* em conjuntos para treino, validação e teste.

Na prática foram criados duas funções para este fim, ambos servem para dividir o *dataset out-car*, um divide a totalidade deste em percentagens e o outro divide-o em amostras isoladas (amostras apresentadas na secção 3.2.2).

Na divisão em percentagens, existe a possibilidade de imagens do mesmo material aparecerem nos três conjuntos: treino, validação e teste. O que torna o método pouco genérico, pois este vai apresentar *overfitting*,



(a) (b)

Figura 55: (a) Imagem original captada, (b) Máscara criada a partir da imagem original com o processo de *Ground Truth Labeler*, onde é possível inspeccionar as diferentes classes. Cada classe apresenta como valor de pixel a atribuição apresentada na Tabela 2 e o *background* apresenta o valor 0.

que é o termo dado ao processo de quando um dado modelo se ajusta muito bem ao *dataset* treinado, mas na presença de outros exemplos mostra uma deteção deficiente de novos resultados. Esta divisão do *dataset* em percentagens, serve para provar o conceito de inspecção de linha, muitas vezes necessário em fábricas que estão constantemente a produzir os mesmos produtos. Esta abordagem, pode ser muito eficaz na deteção de danos desde que o leque de amostras a analisar não varie nem no tipo de danos que podem ocorrer, nem no material para o qual o método foi treinado. Este fenómeno acontece facilmente em linhas de inspecção, nas quais são produzidos os mesmos produtos, estando todos estes sujeitos aos mesmos processos e confinados ao mesmo ambiente de produção, o que garante que os danos de interesse a inspeccionar sejam previsíveis. Em suma, perante uma situação destas, pode considerar-se o uso deste tipo de abordagem uma ótima solução, bastando para isso organizar um *dataset* nas condições anteriormente referidas, e não esperar que um modelo treinado nesta condição seja capaz de decifrar danos diferentes dos já definidos.

O script SplitDatsetV1 (Função SplitDatasetV1 - Anexos A) divide o *dataset out-car* em percentagens, consistindo numa função à qual são passados cinco parâmetros: percentagem de treino; percentagem da validação, caminho absoluto das *features*, caminho absoluto das *labels* e caminho pretendido de *output*. Esta função percorre todo o *dataset* a partir das informações de *featurepath* e *labelpath*. De seguida, procede à randomização dos elementos consoante as percentagens escolhidas para o treino e validação, sendo a percentagem de teste consequentemente: teste=1−(treino + validação). Seguidamente, cria três pastas dentro do *outputpath*: *Train*, *Valid* e *Test*, e dentro de cada uma delas mais duas: *Features* e *Labels*. Em suma, recorrendo à função obtém-se o *dataset* devidamente dividido para uma qualquer percentagem escolhida.

Na divisão em amostras o objetivo consiste em efectuar a divisão total das amostras para cada conjunto: treino, validação e teste. Ao proceder a esta divisão o método torna-se genérico, porque irá proceder ao treino com um conjunto de amostras que não estarão presentes na validação e teste. Este procedimento confere ao método uma generalidade na deteção de danos, preparando-o para se debater com sucesso na classificação de qualquer amostra, reduzindo assim a probabilidade de *overfitting*.

O script SplitDatasetV2 (Função SplitDatasetV2 - Anexos A) divide o *dataset out-car* em amostras, consistindo numa função ao qual são passados seis parâmetros: amostras para treino, amostras para validação,

amostras para teste, caminho absoluto das *features*, caminho absoluto das *labels* e caminho pretendido de *output*. Esta função percorre as amostras que são passadas nos parâmetros da função para cada conjunto: treino, teste e validação, e procede à divisão dessas amostras por pastas. Criando para isso, três pastas dentro do *outputpath*, que são: *Train*, *Valid* e *Test*, e dentro de cada uma delas mais duas: *Features* e *Labels*. Em suma, recorrendo a esta função obtém-se o *dataset* devidamente dividido para os conjuntos de amostras à escolha.

3.2.5 *Conversão do dataset para Object Detection e Semantic Segmentation*

Para treinar os métodos de *Semantic Segmentation* basta recorrer ao processo anteriormente descrito, ou seja, ter acesso às imagens originais do *dataset* (*features*), e à correspondente máscara com o *pixel label* atribuído às classes (*labels*). Por isto, depois de possuir o *dataset* convenientemente dividido segundo uma das abordagens apresentadas na secção 3.2.4, pode surgir a necessidade do redimensionamento do tamanho das imagens para posteriormente serem usadas nos métodos de *Semantic Segmentation*. Neste sentido, foi desenvolvida uma função que serve essencialmente para redimensionar as *features* e *labels*. A função desenvolvida para isso denomina-se *TrainSEG* (ver Função TrainSEG - Anexos A), esta função acede inicialmente ao *dataset* a partir do *datasetinputpath*, onde encontra as *features* e as *labels* que precisam ser redimensionadas para a resolução de *Sfactor*, por último grava as *features* e *labels* já devidamente redimensionadas no *datasetoutputpath*.
Em contrapartida, para realizar o treino de métodos de OD é necessária a presença de uma *bounding box* em torno de cada objeto presente nas *labels*, na Figura 55 (b) é notória a sua falta.
Para resolver este problema, e partindo do algoritmo genérico para métodos de OD (Algoritmo 1), é explicado o cálculo das *bounding boxes* no Algoritmo 2, em Anexos B. Estes dois algoritmos apresentam duas funções, que são: encontrar as *bounding boxes* e proceder ao redimensionamento das *features* e *labels* caso seja pretendido. Concluindo, o desenvolvimento destes algoritmos preparam o *dataset* de modo a serem usados por um qualquer método de OD. O resultado destes algoritmos é apresentado na Figura 56.

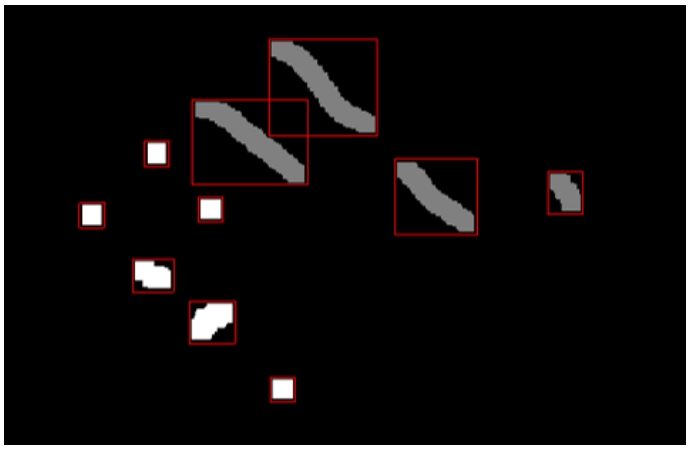


Figura 56: Representação de uma *bounding box* para cada classe.

As *bounding boxes* encontradas nas imagens através do Algoritmo 2 são gravadas num ficheiro *.mat* na forma de uma tabela. A tabela, exemplificada na Figura 57, é composta por colunas e linhas: sendo o número

de colunas igual ao número de classes mais um (coluna que serve para descriminar o caminho absoluto das imagens), e o número de linhas é igual ao número de imagens analisadas. As *bounding boxes* são dadas na forma de 4 coordenadas $[x, y, width, height]$, em que as primeiras duas (x, y) dizem respeito às coordenadas do canto superior esquerdo, e as últimas duas à largura e altura, respetivamente.

	1 image	2 Corte	3 Furo	4 Desgaste	5 Queimadura
1	"C:\Users\Sandra Dixe\Deskt...	[]	[2.5855e+03,949.5000,245,245]	[]	[]
2	"C:\Users\Sandra Dixe\Deskt...	[]	[2.7285e+03,1.0715e+03,267,281]	[]	[]
3	"C:\Users\Sandra Dixe\Deskt...	[]	[1.7345e+03,528.5000,158,166]	[]	[]
4	"C:\Users\Sandra Dixe\Deskt...	[]	[]	[]	[]
5	"C:\Users\Sandra Dixe\Deskt...	[]	[]	[]	[]
6	"C:\Users\Sandra Dixe\Deskt...	[]	[]	[]	[]
7	"C:\Users\Sandra Dixe\Deskt...	[]	[1.3685e+03,861.5000,552,219]	[]	[]
8	"C:\Users\Sandra Dixe\Deskt...	[]	[1.2865e+03,786.5000,634,294]	[]	[]
9	"C:\Users\Sandra Dixe\Deskt...	[]	[331.5000,850.5000,924,1022]	[]	[]
10	"C:\Users\Sandra Dixe\Deskt...	[]	*4x4 double*	[]	[]
11	"C:\Users\Sandra Dixe\Deskt...	[]	[0.5000,823.5000,154,257;0.5000,464.5000,297,616]	[]	[]
12	"C:\Users\Sandra Dixe\Deskt...	[]	*3x4 double*	[]	[]
13	"C:\Users\Sandra Dixe\Deskt...	[]	[2.0275e+03,649.5000,709,171]	[]	[]

Figura 57: Forma de apresentação das *bounding boxes* para o *dataset out-car*.

3.3 Dataset in-car

3.3.1 *Geração de dataset in-car*

Na geração do *dataset in-car* foram recolhidas imagens do interior de carros disponíveis em sucatas e stands de automóveis usados. Para a recolha das imagens em cada carro recorreu-se a dois sensores, apresentando estes resoluções de: 3264x2448 e 1920x1080. Os sensores apresentam ainda campos de visão *ultrawide* (superior a 110°).

A configuração de inspecção usada em cada carro para a geração do *dataset in-car* é a apresentada na Figura 58, onde as posições especificas de cada sensor são representadas por P1 a P9, sendo que de P1 a P8 a câmara é posicionada numa perspectiva descendente e P9 uma perspectiva ascendente.

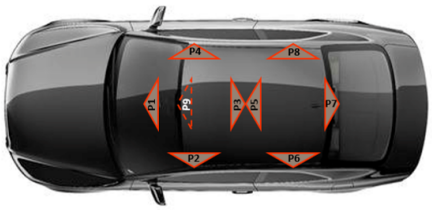


Figura 58: Configuração de inspecção usada em cada carro para a geração do *dataset in-car*.

Nas posições P1 e P5 são captadas imagens em 3 orientações verticais distintas. As perspectivas de todas as posições, P1 a P9, estão descriminadas na Figura 59.

No total, cada sensor fornece 13 imagens por carro, em 78 carros, o *dataset in-car* é composto por um total de 1861 imagens. Desta totalidade, 8 são carros semi-novos encontrados em stands (sem danos) e os restantes 70 são carros de sucata (com danos).

No sentido de se garantir a geração, gestão e exploração do *dataset*, foi definido o pipeline apresentado na

Figura 59: Exemplo das perspectivas das posições de P1 a P9 captadas no interior de cada carro.

Figura 60.

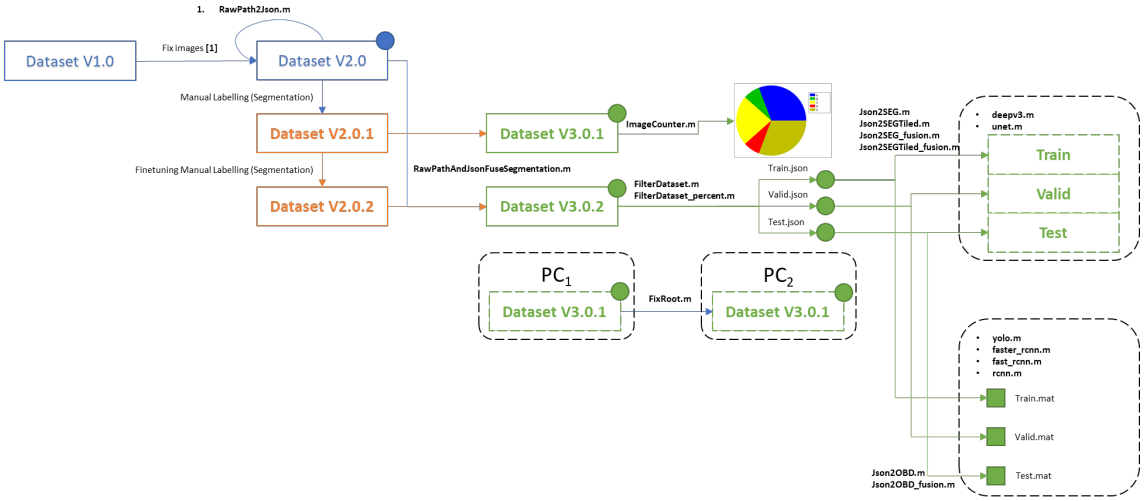


Figura 60: Pipeline de geração e gestão do *dataset in-car*.

A versão Dataset V1.0 é a pasta que reúne as imagens inicialmente capturadas organizadas por carro, as imagens de cada carro estão contidas numa pasta denominada CAR000X sendo o X o número atribuído ao respectivo carro, dentro de cada uma das pastas do carro existem mais 9 pastas, uma para cada posição do sensor (P1 a P9). A versão Dataset V2.0 diz respeito ao processo de filtragem de algumas imagens da versão V1.0, (i.e. apagar imagens desfocadas e corrigir rotações), esta versão passa também pelo algoritmo RawPath2Json.m, onde são guardadas as informações relativas a cada carro, como marca, cor, ano, modelo etc, num ficheiro JSON. Dataset V2.0.1 é o primeiro processo de *labeling* atribuído ao *dataset* (i.e. Manual Labeling (Segmentation)), a versão Dataset V2.0.2 diz respeito ao segundo processo de *labelling* a que o *dataset* foi sujeito (i.e. Finetuning Manual Labeling (Segmentation)), este segundo processo veio dar alguma refinação e definição ao primeiro processo de *labelling* que foi efetuado de maneira mais grosseira, podendo prejudicar o desempenho futuro dos algoritmos de deteção de danos. As versões Dataset V3.0.1 e V3.0.2 surgem da fusão das informações relativas de cada carro e informações do processo de *labeling* de Segmentação no ficheiro JSON através do algoritmo RawPathAndJsonFuseSegmentation.m. O script ImageCounter.m fornece informações da distribuição do *dataset*, como: número de imagens por marca, modelo, cor do interior do carro, número de pixeis de cada classe e percentagem de cada classe contida na totalidade do *dataset* (i.e. indicado pela circunferência preenchida). FilterDataset.m e FilterDataset_percent.m permitem criar versões JSON filtradas, i.e. treino, validação e teste. Todos os JSONs, filtrados ou não, poderão ser utilizados nos métodos de OD e *Semantic Segmentation*. O algoritmo FixRoot.m serve unicamente para actualizar a informação do JSON relativa aos *paths* de acesso ao *dataset*, caso seja necessário.

A organização do *dataset* anteriormente referida, é então gravada numa pasta denominada de Dataset V1.0, versão inicial do *dataset in-car*. No seu interior, as imagens de cada carro são organizadas por pastas denominadas CAR000X sendo o X o número atribuído a cada carro, dentro de cada uma destas pastas são criadas mais 9 pastas, uma para cada posição do sensor de captura (P1 a P9), desta forma obtêm-se as imagens de cada carro devidamente organizadas.

Dentro da pasta Dataset V1.0 é também gravada: uma imagem do exterior de cada carro (para facilitar a sua identificação); e um documento Excel, sendo a finalidade deste a de guardar de forma simples e compacta todas as informações de cada carro, i.e. número atribuído ao carro, marca, modelo, ano, cor e combustível. Com esta organização, está se a atribuir ao *dataset in-car* uma maior reutilização para que este possa ser usado em trabalhos de outros alunos e também em trabalhos futuros.

Depois de organizadas todas as pastas é necessário confirmar todas as imagens capturadas (i.e. Fix images), verificando e corrigindo as suas rotações. Esta etapa será responsável por criar a versão Dataset V2.0.

No sentido de possibilitar a reutilização do *dataset*, as informações do *dataset* (i.e. estrutura, labels, endereços, etc) foram gravadas em formato *JavaScript Object Notation (JSON)*, através do script RawPath2Json.m. Este formato é independente da linguagem de programação, sendo este versátil e compatível com múltiplas linguagens.

O *dataset in-car* passou por várias etapas até chegar à sua versão final, essas etapas estão ilustradas e explicadas na Figura 61. No que diz respeito ao processo de atribuição de *labeling* ao nível do pixel repetiu-se o processo apresentado na secção 3.2.3, este é realizado recorrendo à aplicação do MATLAB *Ground Truth Labeler*, considerando 5 classes (Tabela 3).

Tabela 3: ID atribuido a cada classe do *dataset in-car*.

Nome Classe	Pixel Label ID
DMG_CUT	1
DMG_WEAR	2
DMG_BROKEN	3
STAIN	4
DIRT	5

O processo de *labeling* foi realizado manualmente para as 1861 imagens, o que resultou no Dataset V2.0.1 (i.e. Manual Labeling (Segmentation)). Depois de analisar o *labeling* anterior, concluiu-se que seria boa ideia efectuar o seu refinamento/aprimoramento, ou seja, obter mais perfeccionismo e detalhe na discriminação das classes ao nível do pixel, resultando na versão Dataset V2.0.2 (i.e. Finetuning Manual Labeling (Segmentation)). Na Figura 62, é apresentado um exemplo que traduz a evolução deste processo.

As versões Dataset V3.0.1 e V3.0.2 surgem através do script RawPathAndJsonFuseSegmentation.m, este algoritmo tem como objetivo fundir o ficheiro JSON, no qual foram atribuídas as informações relativas a cada carro de forma manual, com as pastas de resultantes do processo de *labeling* de Segmentação, atribuição da classe ao nível do pixel. As versões Dataset V3.0.1 e V3.0.2 passaram ainda pelo script ImageCounter.m que serve para dar informações à cerca da composição do *dataset*, tais como: número de imagens por marca, modelo, cor do interior do carro, etc. Para a avaliação dos métodos de *Semantic Segmentation* fornece ainda, informações como, número de pixeis para cada classe e a percentagem de cada classe contida

na totalidade do *dataset*. Os scripts FilterDataset_percent e FilterDataset.m são similares e apresentam como finalidade filtrar o *dataset in-car* por percentagens ou por restrições mais especificas como por carros, modelo, e marca. Desta filtragem resultam os conjuntos de treino, validação e teste necessários para treinar os métodos de OD e *Semantic Segmentation*.

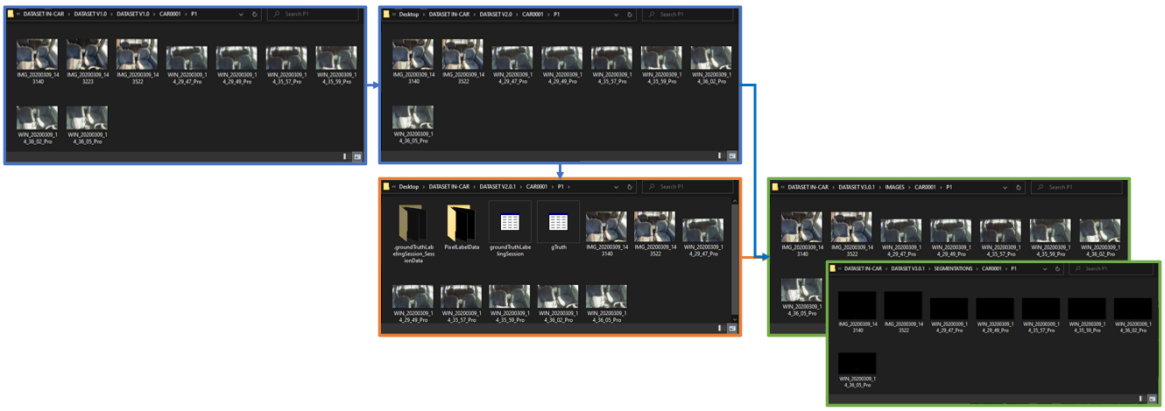


Figura 61: A pasta DATASET V1.0 mais à esquerda na imagem e com contorno a azul corresponde à organização inicial anteriormente descrita, posto isto, foi necessário confirmar a qualidade de todas as imagens (remover imagens desfocadas, corrigir rotações, etc), estas alterações são gravadas na pasta denominada de DATASET V2.0, também com contorno a azul. O DATASET V2.0.1 com contorno a cor de laranja surgiu do resultado da atribuição de *labeling* ao DATASET V2.0, este processo é explicado na secção 3.2.3, e como resultado deste processo, é criada uma pasta denominada *PixelLabelData* com as mascaras criadas para cada imagem e ainda, dois ficheiros *.mat*, um que tem como função guardar a sessão de criação de *labeling* e outro que serve para associar um pixel ID a cada classe criada. A versão final do *dataset* é a pasta DATASET V3.0.1 com contorno a cor verde, esta pasta apresenta duas pastas: IMAGES e SEGMENTATIONS, onde ficam guardadas as imagens capturadas para cada carro e as mascaras correspondentes, respetivamente.

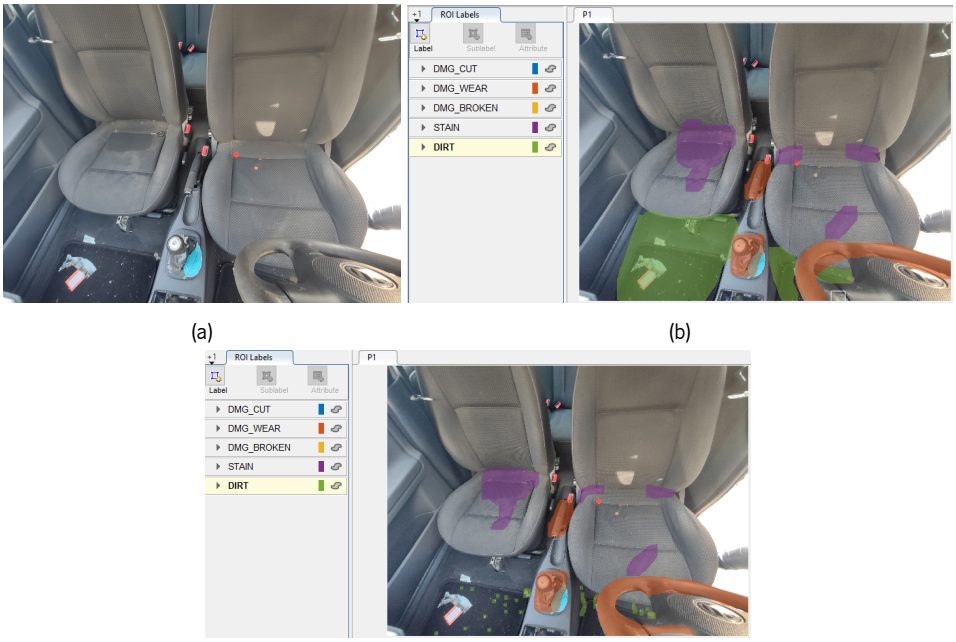
Seguidamente, será apresentada uma explicação para cada script desenvolvido presente na Figura 60. Em anexos A, são apresentadas as propriedades das seguintes funções detalhadamente.

FixRoot (Função FixRoot - Anexos A): tem como função actualizar os caminhos do JSON caso se pretenda aceder a este JSON num computador diferente daquele em que foi criado, ver esquema na Figura 60. A partir do *FixRoot* é possível aceder às informações contidas no JSON em qualquer outro computador, bastando para isso ter uma ligação ao dataset (i.e. disco interno, rede).

RawPath2Json (Função RawPath2Json - Anexos A): esta função gera um ficheiro JSON com as informações da estrutura das sub-pastas do *dataset in-car*. Posteriormente, podem ser preenchidas manualmente as *labels* do JSON. Estas *labels* são informações inerentes aos carros, como: marca, cor combustível, ano, etc.

ImagesCounter (Função ImagesCounter - Anexos A): esta função conta o número de imagens do *dataset* por: Marca, Modelo, Tipo, Cor dos assentos, cor dos plásticos presentes no interior e cor do teto. Para a avaliação dos métodos de *Semantic Segmentation* conta o número de pixeis para cada classe e calcula a percentagem de cada classe presente na totalidade do *dataset*. Na Figura 63 estão apresentados dois exemplos, (a) cálculo por marca, e (b) cálculo por *Segmentation*.

FilterDataset (Função FilterDataset - Anexos A): permite gerar um arquivo JSON filtrado, por exemplo criar um JSON para uma dada marca, ou então, para um conjunto de marcas à escolha. Outra hipótese é por



(a) (b)

(c)

Figura 62: Exemplo a partir de uma imagem que traduz a evolução do *labeling*: inicialmente atribui-se um *labeling* "de modo mais grosseiro", e depois procedeu-se ao seu aprimoramento. (a) Imagem original captada, (b) Imagem com atribuição do primeiro *labeling*, (c) Imagem com atribuição do segundo *labeling* (refinamento).

Label	AbsolutImg	PercentImg
renault	244	13,174946
ford	113	6,101511879
opel	395	21,32829374
fiat	379	20,46436285
volvo	41	2,213822894
mercedes	42	2,267818575
bmw	99	5,345572354
citroen	41	2,213822894
chevrolet	21	1,133909287
porsche	64	3,455723542
daewoo	28	1,51187905
subaru	22	1,187904968
nissan	46	2,483801296
vw	30	1,61987041
honda	89	4,805615551
toyota	31	1,673866091
rover	52	2,807775378
lancia	27	1,457883369
mazda	33	1,781857451
alfa romeo	17	0,917926566

Brand | Model | Type | Seats | Plastics | Ceiling | Segmentation

(a)

Class	Pixels	Percentage(%)
DMG_CUT	25604498	0,252313887
DMG_WEAR	89814343	0,885055663
DMG_BROKEN	10712349	0,105562484
STAIN	391430377	3,857264444
DIRT	78307564	0,771664644

Brand | Model | Type | Seats | Plastics | Ceiling | Segmentation

(b)

Figura 63: Exemplo do ficheiro Excel (a) Cálculo do valor absoluto e valor percentual por marca de automóvel, (b) Cálculo do número de pixeis e valor percentual por classe.

exemplo, filtrar por cor de carro, existem várias conjugações possíveis. A filtragem é feita a partir de uma condição AND. Para ter diferentes condições de filtragem (ou seja, AND, OR), a função deve ser executada numa sequência, em que cada execução é uma condição OR, por exemplo, se quisermos um JSON para carros da marca Renault de cor branca, primeiro é necessário fazer uma filtragem por marca, e com o JSON resultante, efectuar uma filtragem por cor branca dentro de todos os carros da marca Renault obtidos da filtragem anterior.

RawPathAndJsonFuseSegmentation (Função RawPathAndJsonFuseSegmentation - Anexos A): tem

como objetivo fundir um ficheiro JSON, e o seu dataset correspondente, contendo as informações relativas a cada carro; e o directório onde se encontra todo o processo de manual labeling de segmentação. No final será obtido um novo dataset com as imagens (features), segmentações (labels) e um ficheiro JSON com a informação do mesmo.

3.3.2 *Conversão de dataset para os métodos de OD e Semantic Segmentation*

Depois de criados os JSONs pretendidos para cada conjunto: treino, teste e validação, recorrendo ao procedimento anteriormente descrito, é então necessário converter os mesmos para um formato compativel com os métodos de avaliação de danos: *Semantic Segmentation* e OD. Na Figura 60, é apresentado um esquema que exemplifica o processo que os JSONs de cada conjunto passam até poderem ser usados num dado método de avaliação de danos. A seguir é apresentada uma breve explicação de cada script, em Anexos B é apresentada uma explicação mais detalhada de cada um.

Json2OBD (Função Json2OBD - Anexos B): este converte um ficheiro JSON padrão (completo ou filtrado) num arquivo *.mat* para métodos de OD (formato similar à Figura 57).

Json2OBD_fusion (Função Json2OBD_fusion - Anexos B): esta converte um ficheiro JSON padrão (completo ou filtrado) num arquivo *.mat* para ser usado em métodos de OD, possibilitando ainda a fusão de classes. Por exemplo, é possível efectuar a fusão de todas as classes de modo a ser criada uma classe geral, esta super classe pode ser criada com o objetivo de diferenciar o material bom do danificado no interior dos veículos, mas não discrimina os diferentes tipos de danos encontrados nas partes danificadas. A partir deste script podem ser feitas todas as conjugações de fusões de classes possíveis, desde que faça sentido para posterior avaliação nos respectivos métodos de OD. Na Figura 64, é possível observar o conteúdo do ficheiro *.mat* resultante da fusão das classes DMG_CUT, DMG_WEAR e DMG_BROKEN em DAMAGE, ver Tabela 3.

115	"C:\Users\Sandra Dixe\Desktop\...	[]	[1.0715e+03,382.5000,78,73;712.5000,272.5000,206,106]	*5x4 double*
116	"C:\Users\Sandra Dixe\Desktop\...	[878.5000,469.5000,149,51]	[836.5000,850.5000,92,140]	*5x4 double*
117	"C:\Users\Sandra Dixe\Desktop\...	[893.5000,459.5000,146,42]	[861.5000,843.5000,83,108]	*5x4 double*
118	"C:\Users\Sandra Dixe\Desktop\...	[1.0335e+03,736.5000,33,33;952.5000,513.5000,196,63]	[]	*4x4 double*
119	"C:\Users\Sandra Dixe\Desktop\...	[1.6845e+03,909.5000,173,171;50.5000,934.5000,159,146]	[]	[1.0205e+03,773.5000,266,65]
120	"C:\Users\Sandra Dixe\Desktop\...	[1.8385e+03,815.5000,82,71;16.5000,951.5000,98,123]	[]	[1.0855e+03,855.5000,128,82]
121	"C:\Users\Sandra Dixe\Desktop\...	[]	[]	[]
122	"C:\Users\Sandra Dixe\Desktop\...	[894.5000,169.5000,161,65]	*3x4 double*	*4x4 double*
123	"C:\Users\Sandra Dixe\Desktop\...	[775.5000,509.5000,204,71]	[]	[1.4935e+03,798.5000,59,59;1.3645e+03,891.5000,59,59]
124	"C:\Users\Sandra Dixe\Desktop\...	[]	[]	[1.2885e+03,0.5000,632,1080]
125	"C:\Users\Sandra Dixe\Desktop\...	[]	[440.5000,822.5000,171,104]	[1.5225e+03,355.5000,398,725]
126	"C:\Users\Sandra Dixe\Desktop\...	[]	[585.5000,887.5000,135,79]	[1.6295e+03,451.5000,291,629]
127	"C:\Users\Sandra Dixe\Desktop\...	[]	*4x4 double*	[0.5000,780.5000,581,300]
128	"C:\Users\Sandra Dixe\Desktop\...	[]	[1.4115e+03,76.5000,104,87;1.2975e+03,67.5000,68,59]	[]
129	"C:\Users\Sandra Dixe\Desktop\...	*3x4 double*	[]	[746.5000,459.5000,137,90]
130	"C:\Users\Sandra Dixe\Desktop\...	[153.5000,389.5000,427,89]	[1.6615e+03,961.5000,88,68;517.5000,261.5000,188,171]	[1.0425e+03,912.5000,275,162;969.5000,1.0265e+03,59,54]

Figura 64: Ficheiro *.mat* resultando do exemplo dado para a função Json2OBD_fusion.

Json2SEG (Função Json2SEG - Anexos B): este converte um ficheiro JSON padrão (completo ou filtrado) numa pasta com as SEGMENTATIONS e as IMAGES extraídas, todas as imagens são redimensionadas por um sfactor. Processo igual ao apresentado na secção 3.2.5 para métodos de *Semantic Segmentation*, sendo que no *dataset in-car* este processo é realizado a partir de um JSON anteriormente criado. Com o intuito de realizar mais testes nos modelos de *Semantic Segmentation* foi ainda criado um script denominado **Json2SEGTiled**, esta função consegue redimensionar e converter as SEGMENTATIONS e as IMAGES em Tiles, isto é, as imagens do dataset são redimensionadas segundo sfactor e são divididas segundo um factor ntiles (exemplo na Figura 72).

Json2SEG_fusion (Função Json2SEG_fusion - Anexos B): converte um ficheiro JSON padrão (completo ou filtrado) numa pasta com as SEGMENTATIONS e as IMAGES extraídas, todas as imagens podem ser redimensionadas por um sfactor opcional. Com a adição do parâmetro *classesfusion* é possível efectuar uma fusão das classes apresentadas na Tabela 3. Esta função procura os Pixel Labels IDs das classes a serem fundidas nas SEGMENTATIONS e substitui cada um deles por um PixelID comum para todas as classes. No exemplo apresentado abaixo na função *Json2SEG_fusion*, os PixelIDs das classes DMG_CUT, DMG_WEAR e DMG_BROKEN, ver Tabela 3, foram substituídos por um PixelID comum formando assim, a classe geral DAMAGE. Com o intuito de realizar mais testes nos modelos de *Semantic Segmentation* com fusão de classes, foi ainda criado um script denominado **Json2SEGTiled_fusion**, esta função consegue redimensionar e converter as SEGMENTATIONS e as IMAGES com as classes fundidas segundo a atribuiação a *classesfusion* em Tiles, isto é, as imagens do dataset são redimensionadas segundo sfactor e são dividas segundo um factor ntiles (exemplo na Figura 72).

4

DESENVOLVIMENTO DE ALGORITMOS PARA DETEÇÃO DE DANOS OUT-CAR

Neste capítulo é apresentada uma explicação dos algoritmos desenvolvidos para a deteção de danos no interior de um carro. Todos os algoritmos desenvolvidos foram adaptados do Estado de arte, apresentado no Capítulo 2. O objetivo fulcral é fazer uma pré-seleção de algoritmos para posteriormente avaliar no *dataset in-car*. O capítulo está assim dividido em duas partes.
Numa primeira parte será efectuada uma explicação detalhada da ferramenta MATLAB onde foram desenvolvidos/adaptados os algoritmos, e numa segunda parte serão apresentados os resultados do desempenho dos algoritmos obtidos no *dataset out-car*. A partir da análise dos resultados obtidos, será apresentada uma breve conclusão de quais os melhores métodos para deteção de danos no interior de um carro recorrendo ao *dataset out-car* anteriormente desenvolvido. Os algoritmos que apresentarem os melhores desempenhos serão avaliados numa fase posterior, no *dataset in-car*. Todos os ensaios foram efectuados num servidor com um processador Intel(R) Xeon(R) Gold 6140 CPU 2.30Ghz, memória RAM de 128GB e GPU de computação NVIDIA Tesla V100-PCIE-16GB.

4.1 Implementação de algoritmos para deteção de danos

No Estado de arte apresentado na Capítulo 2, são revistos vários artigos com algoritmos eficazes na deteção de danos. Uma boa opção para resolver o problema proposto nesta dissertação pode estar no uso de abordagens de DL, nomeadamente algoritmos de OD e *Semantic Segmentation*. A ferramenta MATLAB 2019b disponibiliza uma *Toolbox de Deep Learning* que contém uma *Toolbox de Computer Vision*, onde são disponibilizados exemplos de algoritmos de OD e *Semantic Segmentation*. Os algoritmos implementados estão discriminados na Tabela 4.

Tabela 4: Métodos de Deep Learning desenvolvidos no MATLAB para avaliação no *dataset out-car.*

Object Detection	- YOLOv2
Semantic Segmentation	- DeepLabv3+ - U-Net

Considerando a necessidade de efectuar vários ensaios aos diferentes métodos, preservando as mesmas condições independentemente do método. Surge a necessidade de automatizar o processo de treino, validação e teste para cada um dos métodos, garantindo uma maior transversalidade dos parâmetros de treino. A partir

desta abordagem, torna-se muito mais fácil gerir ensaios comuns para todos os métodos, deixando assim, de existir a necessidade de adaptar os parâmetros para cada método individualmente. Resultando assim, numa gestão e preparação muito mais fácil dos conjuntos de treinos/ensaios para posterior comparação entre eles. Assim sendo, foi criada uma estrutura quase comum para os algoritmos da Tabela 4. Devido às especificidades dos algoritmos de OD e *Semantic Segmentation* não foi possível traduzi-los em parâmetros de entrada totalmente iguais entre si (i.e. nanchors).
Os parâmetros de entradas para cada método podem ser vistos nos itens 1 e 2 e a sua explicação é apresentada em Anexos B nas Funções Semantic_Segmentation e Object_Detection, respetivamente. É possível verificar que a ordem dos parâmetros de entrada para cada algoritmo se mantém entre métodos, isto é importante para facilitar a criação de um script de ensaios, para extração de uma comparação entre métodos.

1. **Semantic Segmentation** (Função Semantic_Segmentation automatizada com o Algoritmo 1 - Anexo B)
   Semantic_Segmantation(datasetpath,modelpath,lr,solver,epochs,batch,istrain,isresume,istest)

2. **Object Detection** (Função Object_Detection automatizada com o Algoritmo 1 - Anexo B)
   Object_Detection(datasetpath,modelpath,lr,solver,epochs,batch,nanchors,isanchors,isplotanchors,istrain, isresume,istest)

O Algoritmo 1 apresentado em Anexos B é referente aos modelos de *Semantic Segmentation* e OD (i.e, item 1 e 2), este é transversal a todos os modelos no sentido de permitir uma maior automação no desenvolvimento dos mesmos, no entanto nos modelos de OD existe a adição de uma nova etapa em (2) que é o cálculo das anchor boxes que é inerente a estes modelos (Algoritmo 2 - Anexos B).
O YOLOv2, não obstante das diferenças do modelo comparativamente com os Segmentadores, necessita de pré-processar as *anchor boxes* como é referenciado pelos autores no Capitulo 2. O método de OD YOLOv2 necessita de *anchor boxes*. A forma, a escala e o número de *anchor boxes* afectam diretamente a eficiência e a precisão deste. As *anchor boxes* são estimadas a partir do *dataset* de treino. A função *estimateAnchorBoxes* do MATLAB calcula a *mean intersection-over-union* (IoU) *distance metric*.
A escolha do número de *anchor boxes* é um hiperparâmetro de treino que requer uma seleção cuidadosa usando análise empírica. A função *estimateAnchorBoxes* usa um algoritmo de *k- mean clustering* com a métrica de IoU *distance* para calcular a sobreposição das *anchor boxes* com as *bounding boxes* do *dataset* de treino, o gráfico resultante da função esta apresentado na Figura 65.
Um valor de *mean* IoU maior que 0.5 garante que as *anchor boxes* se sobreponham bem às *bounding boxes* do *dataset* de treino. Com o aumento do número de *anchors* o valor de *mean* IoU melhora significativamente. No entanto, o uso de muitas *anchor boxes* num método de OD aumenta o custo da computacional e leva ao fenómeno de *overfitting*, o que resulta num mau desempenho por parte do método de OD.
Tanto os métodos de OD como de *Semantic Segmentation* necessitam de redes pré-treinadas (*backbones*). O MATLAB possibilita a visualização das arquiteturas de DL usando a função *analyzeNetwork*. A *analyzeNetwork* exibe uma visualização interactiva da arquitetura de uma dada rede, detectando erros e problemas na rede e fornecendo informações detalhadas sobre as camadas da rede, ver Figura 66. Caso seja necessário um maior controlo sobre a arquitetura do YOLOv2 ou qualquer outra arquitetura, o *Deep Network Designer* disponibiliza essa flexibilidade, dando hipótese ao utilizador de projectar manualmente uma rede do início ou

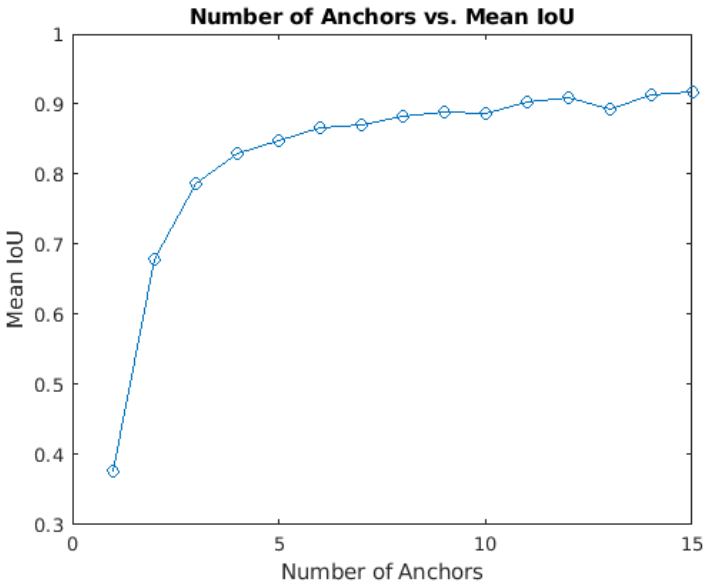


Figura 65: Segundo o *Help Center* do MATLAB para este exemplo, o uso de duas *anchor boxes* resulta num valor de *mean* IoU maior que 0.65, e o uso de mais de 7 *anchor boxes* produz um aumento pouco significativo de *mean* IoU. Em suma, é importante treinar e avaliar os vários métodos de OD usando valores de *anchor boxes* entre 2 e 6. Esta análise empírica ajuda a determinar o número de *anchor boxes* necessárias para atender a requisitos de desempenho, como velocidade de detecção e precisão.

então fazer alterações em redes já existentes, ver Figura 67. Esta ferramenta possibilita ao utilizador efectuar muitos testes e tirar muitas conclusões.

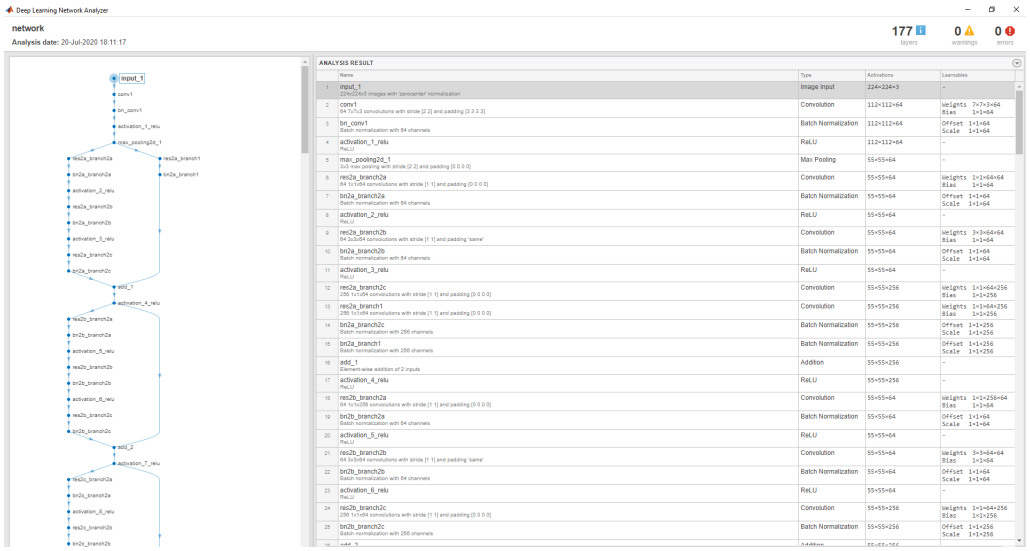
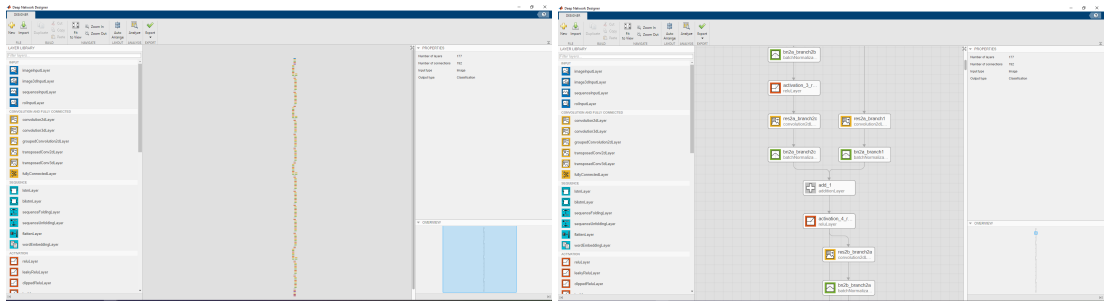


Figura 66: Exemplo da arquitetura da rede ResNet-50 fornecida pela função *analyzeNetwork* disponivel no Deep Network Designer do Deep Learning Toolbox. Esta ferramenta permite realizar uma interpretação pormenorizada da rede.

Outro método importante na melhoria da precisão da rede é o *Data Augmentation* tanto para métodos de OD como de *Semantic Segmentation*, esta técnica transforma aleatoriamente o *dataset* original durante o treino ao inverter aleatoriamente a imagem e as *box labels* horizontalmente. Ao usar *Data Augmentation*



(a) (b)

Figura 67: Ferramenta *Deep Network Designer* do MATLAB que permite projectar arquiteturas de redes a partir do zero, ou então, alterar arquiteturas já existentes. É possível apagar operações a partir de blocos ou então, acrescentar novas operações, tudo isto, de uma forma gráfica. (a) Arquitetura da rede ResNet-50, visão geral (Zoom out), (b) Arquitetura da rede ResNet-50 com Zoom in, onde existe a possibilidade de ver ou alterar todas as operações constituintes desta arquitetura.

está-se a adicionar mais variedade ao *dataset* de treino sem a necessidade de aumentar o número de amostras de treino e correspondentes amostras com *labeling*. Esta transformação não é aplicada aos *datasets* de teste e validação. A ideia é que os estes *datasets* sejam representativos dos dados originais e não sejam modificados para que a avaliação seja imparcial.

A função *trainingOptions* serve para especificar opções de treino para a rede, tanto para métodos de OD como de *Semantic Segmentation*, ver exemplo na Figura 68. No treino de todos os algoritmos foi usado um *learning rate* adaptável, ou seja, que se ajusta automaticamente para obter um melhor desempenho. Na função *trainingOptions* é definido um *LearnRateDropPeriod* que é o número de *epochs* ás quais o *learning rate* diminui segundo o factor definido em *LearnRateDropFactor* do valor definido originalmente. A opção *CheckpointPath* permite salvar detectores parcialmente treinados durante o processo de treino. Se o treino for interrompido, por alguma falha na energia ou no sistema, torna-se possível através desta opção retomar o treino a partir do último ponto de verificação salvo.

```
options = trainingOptions(solver, ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropPeriod',20,...
    'LearnRateDropFactor',0.8,...
    'InitialLearnRate',lr, ...
    'L2Regularization',0.005, ...
   'MiniBatchSize', batch, ....
   'InitialLearnRate',lr, ...
   'MaxEpochs', epochs,...
   'CheckpointPath', modelpath, ...
   'ExecutionEnvironment', 'gpu');
```

Figura 68: Exemplo da função *trainingOptions*, que serve para especificar opções de treino tanto para os métodos de OD como para os de *Semantic Segmentation*.

O tempo de cada treino depende essencialmente do hardware disponível. Depois de concluído o treino torna-se possível proceder à avaliação dos métodos. Esta avaliação é realizada no *dataset* de teste e permite-nos avaliar o desempenho do treino anteriormente realizado. A *Computer Vision Toolbox* fornece funções de avaliação para métodos de OD e *Semantic Segmentation*, que medem métricas comuns, como *average*

precision através da função *evaluateDetectionPrecision* e taxas médias de erros. A *average precision* fornece um único número que incorpora a capacidade do método fazer classificações corretas *precision* e a capacidade de encontrar todos os objetos ou classes relevantes *recall*.
Em suma, como pode ser constatado na explicação acima, os algoritmos dos métodos presentes na Tabela 4, são bastante similares na sua generalidade.

4.2 Avaliação e seleção dos algoritmos no dataset out-car

Uma vez criado o *dataset out-car*, ver esquema na Figura 69, e concluído o desenvolvimento do algoritmo de OD e os de *Semantic Segmentation* propostos na Tabela 4, é necessário proceder à avaliação destes algoritmos, com o intuito de perceber qual é a abordagem que melhor se adapta na avaliação deste tipo de danos. O objetivo desta secção é fazer uma pré-selecção de algoritmos para posteriormente serem avaliados no *dataset in-car*. Os algoritmos inicialmente avaliados no *dataset out-car* são os apresentados na Tabela 5, em que OEV1 diz respeito ao método de OD YOLOv2 com *backbone* resnet50 e o método OEV2 diz respeito ao método de *Semantic Segmentation* DeepLabV3+ com *backbone* resnet18.

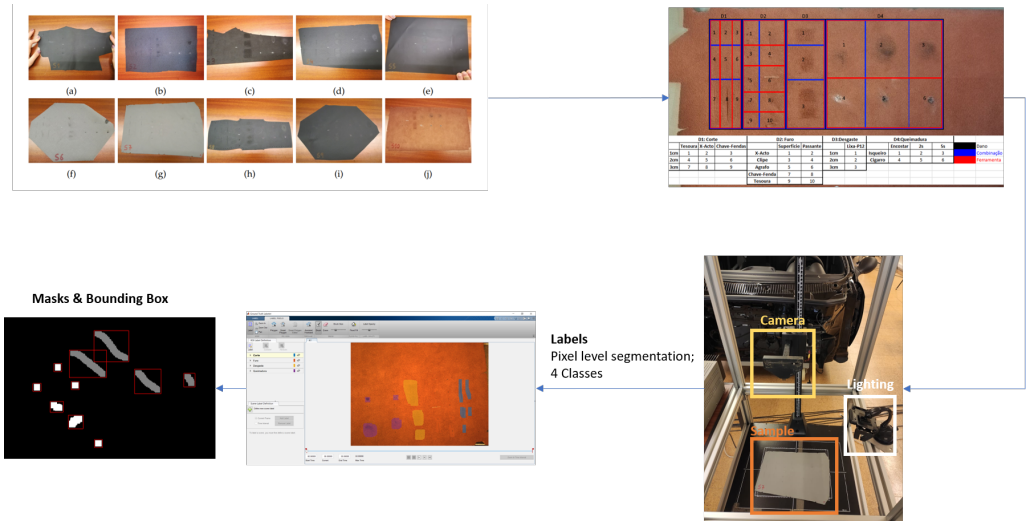


Figura 69: Esquema de criação do *dataset out-car*, 10 amostras de materiais comummente utilizados no fabrico de interiores de automóveis são expostos ao *template* de danos apresentado na imagem (Corte, Furo, Desgaste e Queimadura). Posto isto, são recolhidas um total de 1600 fotografias das amostras com diferentes ângulos e perspectivas, onde é realizado o processo de *labeling* (atribuição das classes ao nível do pixel). Todo o processo é explicado pormenorizadamente na secção 3.2.3.

4.2.1 *Object Detection vs Semantic Segmentation*

Os métodos OEV1 e OEV2 (Tabela 5) foram testados inicialmente com uma configuração tipicamente Industrial a nível de divisão de *dataset* (Tabela 6), 60% para treino, 20% para validação e 20% para teste, no qual se realizou um estudo de ablação inicial a nível de resolução de entrada de rede, este estudo consistiu em

testar OEV1 e OEV2 com 4 configurações diferentes de rede T1, T2, T3 e T4, com 300x300x3, 500x500x3, 1000x1000x3 e 1200x1600x3, respetivamente (Tabela 7).

Tabela 5: Algoritmos de *Deep Learning* desenvolvidos no MATLAB testados no *dataset out-car*.

OEV1	Object Detection: YOLOv2 (resnet50)
OEV2	Semantic Segmentation: DeepLabV3+ (resnet18)

Tabela 6: Divisões do *dataset out-car* apresentadas e explicadas na Secção 3.2.4.

Industrial	Train = 60%, Valid = 20%, Test = 20%
Challenge	Train = S1/S9/S8/S7/S10 (5/8), Valid = S4 (1/8), Test = S2/S3 (2/8)

Tabela 7: Estudo de ablação ao nível de resoluções de rede de entrada para os métodos OEV1 e OEV2.

	Resolução de entrada
T1	300x300x3
T2	500x500x3
T3	1000x1000x3
T4	1200x1600x3

Os ensaios para OEV1 e OEV2, foram realizados para classificação das 4 classes presentes no *dataset out-car* (corte, furo, desgaste e queimadura), no caso do método OEV2 que efectua classificação ao nível do pixel, o *background* também é uma classe (i.e. bom). Todos os ensaios foram realizados recorrendo a uma configuração similar: 100 *epochs*, otimizador ADAM, 0.001 de *learning rate*, em OEV1 usou-se uma *batch* de 8 para as resoluções 300x300x3 e 500x500x3 e uma *batch* de 2 para as resoluções de 1000x1000x3 e 1200x1600x3, em OEV2 usaram-se *batch* de 16, 8, 2, 2 para as resoluções de 300x300x3, 500x500x3, 1000x1000x3 e 1200x1600x3, respetivamente. Os resultados são apresentados na Tabela 8.
Para os melhores ensaios obtidos em cada método (i.e. OEV1:T3 e OEV2:T4) discriminados a negrito na Tabela 8, foi realizado o mesmo ensaio mudando a configuração de divisão de *dataset out-car* agora para uma divisão Challenge (Tabela 6), comummente encontrada em artigos, nesta configuração o *dataset* é divido por amostras isoladas para cada conjunto: S1/S7/S8/S9/S10 para treino, S4 para validação e S2/S3 para teste. Os resultados obtidos estão descriminados na Tabela 9.

Observando os resultados obtidos para OEV1 e OEV2 (Tabela 9) procedeu-se a uma otimização (estudo de ablação e hiperparâmetros) para cada modelo, apresentada nas subsecções 4.2.2 e 4.2.3 para OEV1 e OEV2, respetivamente.

Tabela 8: Resultados do estudo de ablação efectuado nos métodos OEV1 e OEV2 (divisão Industrial) nível da resolução de entrada de rede, os ensaios que obtiveram a melhor performance estão descriminados a negrito, ensaio T3 (1000x1000x3) para o método OEV1 e T4 (1200x1600x3) para o método OEV2. A performance do método de OD OEV1 é dada pela métrica Precisão Média ($TP_{IoU>=50\%}$), e do método de *Semantic Segmentation* OEV2 é dada pela métrica Exatidão Média. Para os dois métodos é fornecida a métrica obtida na classificação de cada classe e a classificação total do método no respectivo ensaio.

			Precisão Média					
			CORTE	**FURO**	**DESGASTE**	**QUEIMADURA**	**-**	**TOTAL**
Industrial	**OEV1**	T1	0%	0%	0%	0%	-	0%
		T2	0.27%	0.53%	3.43%	1.52%	-	1.44%
		T3	**71.21%**	**68.88%**	**68.05%**	**81.03%**	**-**	**72.29%**
		T4	54.27%	56.11%	39.63%	70.51%	-	55.13%
			Exatidão Média					
			CORTE	**FURO**	**DESGASTE**	**QUEIMADURA**	**BOM**	**TOTAL**
	OEV2	T1	75.74%	71.91%	87.44%	93.94%	92.92%	84.59%
		T2	82.00%	78.46%	86.46%	94.70%	86.47%	85.62%
		T3	87.81%	87.70%	93.16%	91.48%	85.45%	89.11%
		T4	**94.43%**	**97.81%**	**93.50%**	**94.91%**	**92.08%**	**94.55%**

Tabela 9: Resultados obtidos nos métodos OEV1 T3 e OEV2 T4 com divisão de *dataset* Challenge. A performance do método de OD OEV1 é dada pela métrica Precisão Média ($TP_{IoU>=50\%}$), e do método de *Semantic Segmentation* OEV2 é dada pela métrica Exatidão Média. Para os dois métodos é fornecida a métrica obtida na classificação de cada classe e a classificação total do método no respectivo ensaio.

			Precisão Média					
			CORTE	**FURO**	**DESGASTE**	**QUEIMADURA**	**-**	**TOTAL**
Challenge	**OEV1**	T3	0%	0%	0%	0%	-	0%
			Exatidão Média					
			CORTE	**FURO**	**DESGASTE**	**QUEIMADURA**	**BOM**	**TOTAL**
	OEV2	**T4**	**26.42%**	**36.81%**	**24.42%**	**23.85%**	**93.25%**	**40.95%**

4.2.2 *Optimização de Object Detection*

O ensaio OEV1 T3 (Tabela 9) não obteve resultados de Precisão, pelo que se decidiu testar este método com novas resoluções de entrada de rede, T5 e T6, com 896x896x3 e 1344x1344x3, respetivamente, as resoluções de entrada T5 e T6 propostas são múltiplas das apresentadas pelo autor do método (Tabela 10). O método OEV1 foi testado nestas configurações T5 e T6 para as divisões de *dataset* Industrial e Challenge (Tabelas 6). Os resultados são apresentados na Tabela 11.

O método OEV1 na configuração Industrial manteve uma boa performance, não se verificando grandes diferenças entre as configurações T3 e T5 (ver Tabelas 8 e 11), estes foram os melhores ensaios de OEV1 e estão descriminados a negrito. Por outro lado, o método OEV1 na configuração Challenge no ensaio T5 falhou, já com configuração de entrada T6 verificamos uma melhora significativa na classificação de certas classes por parte de OEV1 comparativamente com OEV1 T3 na divisão Challenge (Tabela 9), mas ainda assim, a perfor-

Tabela 10: Estudo de ablação ao nível de resoluções de rede de entrada para os métodos OEV1 e OEV2.

	Resolução de entrada
T5	896x896x3
T6	1344x1344x3

Tabela 11: Resultados obtidos no método OEV1 para as novas configurações de resolução de entrada de rede T5 e T6, para as divisões de *dataset* Industrial e Challenge. A performance do método é apresentada segundo a métrica Precisão Média obtida na classificação de cada classe e na classificação total do método no respectivo ensaio.

			Exatidão Média				
			CORTE	**FURO**	**DESGASTE**	**QUEIMADURA**	**TOTAL**
OEV1	**Industrial**	**T5**	**71.21%**	**68.89%**	**68.05%**	**81.03%**	**72.30%**
		T6	54.28%	56.12%	39.63%	70.51%	55.14%
	Challenge	T5	0%	0%	0%	0%	0%
		T6	14.67%	25.12%	0%	0.64%	13.48%

mance é muito baixa e desequilibrada entre classes, por exemplo, para as classes: desgaste e queimadura, o método não apresenta classificação (Tabela 11). Com base nos factos mencionados anteriormente, decidiu-se abandonar o método de OD OEV1 e prosseguir com o estudo do método de *Semantic Segmentation* OEV2, este mostrou ser bastante equilibrado na classificação das classes, apresentando uma excelente performance de classificação na configuração Industrial e uma performance mais baixa na configuração Challenge como é compreensível, mas apresenta grande equilíbrio nas métricas de classificação para todas as classes. Em suma, este método OEV2 DeepLabV3+ (resnet18) mostra-se ser promissor na classificação deste tipo de classes daí a continuação do seu estudo.

4.2.3 *Optimização de Semantic Segmentation*

Para o método de *Semantic Segmentation* DeepLabV3+ OEV2, será efetuado um novo estudo de ablação a nível de *backbones*, diferentes *backbones* serão testadas para a melhor resolução de entrada obtida em OEV2, que é T4 1200x1600x3 (ver Tabelas 8 e 9). O método de *Semantic Segmentation* U-Net será também testado neste contexto, esta escolha deveu-se ao facto de este ser um método que se mostra muito eficiente e poderoso a nível de classificação em trabalhos do Estado de arte. Os novos ensaios OEV3 a OEV7 estão discriminados na Tabela 12, em que, OEV3 a OEV6 correspondem ao método DeepLabV3+ testando diferentes *backbones* e OEV7 corresponde ao método U-Net.
Nas Tabelas 13 e 14 estão apresentados os resultados obtidos nos métodos OEV3 a OEV7 com resolução de entrada de rede de T4 (1200x1600x3) para as divisões de *dataset* Industrial e Challenge, respetivamente.

Os ensaios que obtiveram as melhores performances foram os ensaios correspondentes ao método DeepLabV3+, OEV2 e OEV4 com *backbone* resnet18 e mobilenetv2 (Divisão Industrial Tabelas 8 e 13, Divisão

Tabela 12: Novos ensaios propostos segundo o estudo de ablação para o método DeepLabV3+ considerando diferentes tipos de *backbones* (OEV3 - OEV6). OEV7 diz respeito ao método de *Semantic Segmentation* U-Net.

	Modelo
OEV3	DeepLabV3+ (resnet50)
OEV4	DeepLabV3+ (mobilenetv2)
OEV5	DeepLabV3+ (xception)
OEV6	DeepLabV3+ (inceptionresnetv2)
OEV7	U-Net

Tabela 13: Resultados obtidos nos métodos de OEV3 a OEV7 com resolução de entrada de rede T4 (1200x1600x3) com divisão de *dataset* Industrial. A performance dos métodos é dada pela métrica Exatidão Média, no qual é fornecida a métrica obtida na classificação de cada classe e a classificação total do método no respectivo ensaio.

			Exatidão Média					
			CORTE	**FURO**	**DESGASTE**	**QUEIMADURA**	**BOM**	**TOTAL**
Industrial	**OEV3**	T4	54.91%	79.01%	68.49%	51.85%	81.51%	67.15%
	OEV4	**T4**	**93.24%**	**97.69%**	**82.15%**	**93.14%**	**92.83%**	**91.81%**
	OEV5	T4	59.74%	90.69%	72.19%	73.16%	76.73%	74.50%
	OEV6	T4	83.93%	76.92%	70.33%	67.15%	95.42%	78.75%
	OEV7	T4	24.30%	27.07%	40.04%	18.65%	53.00%	32.61%

Tabela 14: Resultados obtidos nos métodos de OEV3 a OEV7 com resolução de entrada de rede T4 (1200x1600x3) com divisão de *dataset* Challenge. A performance dos métodos é dada pela métrica Exatidão Média, no qual é fornecida a métrica obtida na classificação de cada classe e a classificação total do método no respectivo ensaio.

			Exatidão Média					
			CORTE	**FURO**	**DESGASTE**	**QUEIMADURA**	**BOM**	**TOTAL**
Challenge	**OEV3**	T4	29.04%	46.06%	14.23%	27.98%	88.81%	41.22%
	OEV4	**T4**	**51.01%**	**48.58%**	**8.49%**	**22.82%**	**96.53%**	**45.49%**
	OEV5	T4	28.70%	31.69%	14.97%	17.33%	82.08%	34.95%
	OEV6	T4	15.87%	30.41%	11.62%	12.15%	86.64%	31.34%
	OEV7	T4	13.53%	13.48%	11.43%	2.15%	81.53%	24.42%

Challenge Tabelas 9 e 14), respetivamente. O OEV2 obteve uma exatidão média total de 94.55% na avaliação Industrial e de 40.95% na avaliação Challenge. O OEV4 obteve uma Exatidão média de 91.91% na avaliação Industrial e 45.49% na avaliação Challenge. O ensaio OEV2 ganhou na divisão Industrial, mas perdeu na divisão Challenge para o OEV4, mas o que se verifica é que o método OEV2 é mais estável, apresenta valores de exatidão média balanceados entre as classes na divisão Challenge apresar, destes valores serem mais baixos. No método OEV4 é possível constatar (Tabela 14) que o ensaio desce consideravelmente o valor da exactidão na classificação da classe desgaste em relação às outras classes, pelo que não é tão balanceado

como o OEV2.

Em suma, o ensaio que mostrou a melhor performance foi o OEV2, que consiste no método de *Semantic Segmentation* com *backbone* resnet18. Como curiosidade decidiu-se proceder à avaliação do melhor método encontrado (i.e. OEV2:T4), mas desta vez treinando-o para classificar todas as classes presentes no *dataset in-car* (corte, furo, desgaste e queimadura) como uma única classe denominada NOK, para isso recorreu-se à função *Json2SEG_fusion*, que efectua a fusão de classes, ou seja estes ensaios vão apenas avaliar se existe dano ou não, classificando o material entre material bom ou danificado. Esta é uma avaliação plausível porque o objetivo desta dissertação é identificar a existência de danos no interior de carros, não sendo apresentada a necessidade de classificar/distinguir o tipo de dano em causa. O OEV2 foi testado segundo a divisão Industrial e Challenge, e o resultado está apresentado na Tabela 15.

Tabela 15: Resultados obtidos na avaliação do método OEV2 na classificação de uma classe dano generalizada NOK. A performance dos ensaios é dada pela métrica Exatidão Média, no qual é fornecida a métrica obtida na classificação de cada classe e a classificação total do método no respectivo ensaio.

			Exatidão Média		
			NOK	**BOM**	**TOTAL**
OEV2.1	**Industrial**	T4	89.14%	92.80%	90.97%
	Challenge	T4	75.28%	73.21%	72.25%

A performance global do método OEV2 na classificação diferenciada das classes é de 94.55% e 40.95%, na avaliação Industrial e Challenge, respetivamente, já na classificação da classe única de dano NOK é de 90.97% e 72.25% na avaliação Industrial e Challenge, respetivamente. Em suma, conclui-se que apesar de a performance baixar ligeiramente na avaliação com divisão Industrial, na divisão Challenge o desempenho é consideravelmente melhor, pelo que pode ser considerada uma boa abordagem na classificação do tipo de dano NOK (classe que engloba todas as classes de dano) em divisão Challenge.

Para perceber de que forma é apresentada a inferência de danos pelos métodos de OD e *Semantic Segmentation* podemos observar o exemplo apresentado na Figura 70. Na Figura 70 (a) é observável uma inferência obtida pelo método OEV1 com resolução de entrada de rede T3, este método de OD YOLOV2 obteve boas métricas de Precisão Média na divisão de *dataset* Industrial, na Figura 70 (b) é observável a inferência obtida pelo método OEV1 com resolução de entrada de rede T6 na divisão *dataset* Challenge, neste ensaio o método apresenta baixos valores percentuais de Precisão Média. Na Figura 70 (c) pode-se observar uma boa inferência de danos através do método de *Semantic Segmentation* OEV2 T4 na divisão de *dataset* Industrial, e na Figura 70 (d) está um exemplo de uma má inferência obtida pelo método OEV2 T4 na divisão de *dataset* Challenge.

4.3 Discussão e conclusão

Neste capitulo são apresentados e explicados os *scripts* usados para o treino dos métodos de OD YOLOv2 e *Semantic Segmentation*: DeepLabv3+ e U-Net. No caso do método YOLOv2 é apresentada uma breve explicação do processo efetuado pelo MATLAB para o cálculo das *anchor boxes*, característica dos métodos

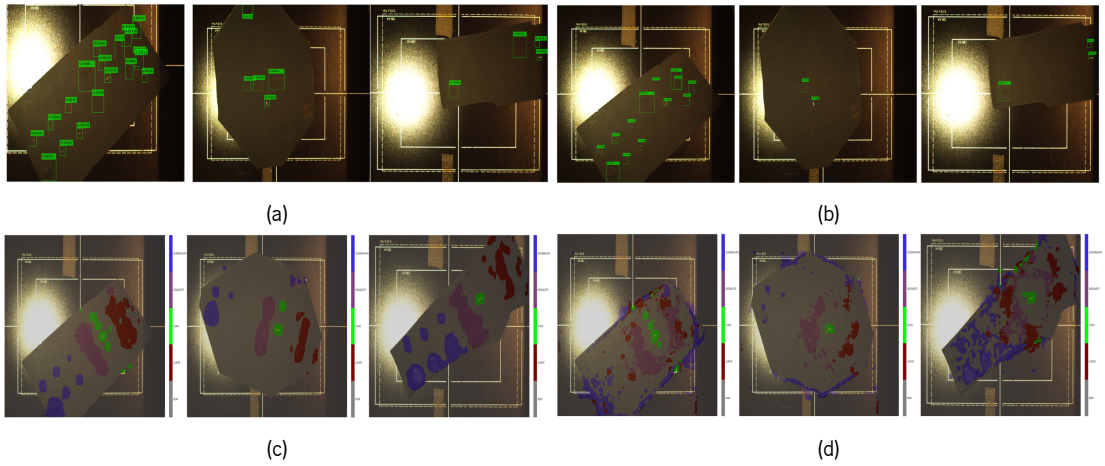


Figura 70: Representação da inferência dos métodos de OD e *Semantic Segmentation*. (a) Exemplo de inferência do método de OD com alto valor de Precisão Média, muitas *bounding boxes* preditas corretamente, (b) Inferência de um método de OD com baixa Precisão Média, poucas *bounding boxes* preditas ou existência de predição incorrecta, (c) Exemplo de inferência do método de *Semantic Segmentation* com alto valor de Exatidão Média, classes ao nível do pixel atribuídas corretamente na imagem, (d) Exemplo de inferência do método de *Semantic Segmentation* com baixa Exatidão Média, classes ao nível do pixel atribuídas incorrectamente, são detectados danos em áreas não existentes.

de OD. É também apresentada uma explicação de como se pode observar e manipular as arquiteturas de rede (*backbones*) recorrendo ao *analyzeNetwork* e *Deep Network Designer*, respetivamente. Também é dado a conhecer um pouco do processo de *Data Augmentation* usado tanto pelos métodos de OD como *Semantic Segmentation*. Por último, são apresentados os parâmetros de interesse para efectuar os treinos dos métodos através da função *trainingOptions* disponibilizada pelo MATLAB e uma breve apresentação das métricas usadas para avaliar cada um dos treinos dos métodos.

Seguidamente foram treinados os métodos no *dataset out-car*, e apresentada a sua avaliação. Conclui-se que, o método de *Semantic Segmentation* OEV2 DeepLabV3+ (resnet18) obteve uma performance bastante promissora na classificação de classes de dano no *dataset out-car*, tanto na abordagem de divisão de *dataset* Industrial como Challenge, pelo que é uma abordagem que será testada seguidamente no *dataset in-car*. O método de *Semantic Segmentation* OEV7 U-Net também será testado e avaliado no contexto de *dataset in-car*, apesar de não ter obtido valores de métricas muito elevados na avaliação do *dataset out-car* (Tabelas 13 e 14), mostra algum equilíbrio na classificação de todas as classes de dano (corte, furo, desgaste e queimadura) e como é um algoritmo que apresenta um bom comportamento em estudos apresentados no Estado de arte, é pertinente o estudo e avaliação da U-Net no contexto da avaliação das classes no interior de um carro (*dataset in-car*).

5

AVALIAÇÃO DOS ALGORITMOS NO INTERIOR DO VEÍCULO

Serão apresentados os resultados obtidos na avaliação dos algoritmos de deteção de danos no *dataset in-car*. Os algoritmos usados na avaliação do *dataset in-car* são os que obtiveram melhores performances no capítulo 4.2. No fim deste capítulo, será ainda apresentada uma breve discussão acerca dos resultados obtidos na resolução da problemática presente nesta dissertação e uma conclusão. Uma versão preliminar da criação do dataset in-car é apresentada nos artigos "**In-Car Damage and Stain Estimation with RGB Images**"e "**In-Car State Classification with RGB Images**".
Todos os ensaios serão efectuados num processador da Intel(R) Xeon(R) Gold 6140 CPU 2.30Ghz, memória RAM de 128GB e GPU de computação NVIDIA Tesla V100-PCIE-16GB.

5.1 Avaliação dos algoritmos seleccionados

O objetivo desta secção é proceder à avaliação dos melhores métodos seleccionados no capítulo 4.2. Os algoritmos selecionados na avaliação do *dataset out-car*, apresentados no capítulo 4.2, são os métodos de *Semantic Segmentation* DeepLabV3+ e U-Net. As redes DeepLabV3+ e U-Net serão treinadas e testadas no *dataset in-car*, segundo duas abordagens de divisão de *dataset*: Industrial e Challenge.
Na divisão Industrial o *dataset* é dividido em percentagem para os conjuntos de treino, validação e teste, esta é uma abordagem típica de inspecção industrial, em que o *dataset* é divido aleatoriamente por percentagens, o que pode facilmente resultar em amostras de imagens do mesmo carro nos três conjuntos de treino. Caso a inspecção de danos seja efectuada sempre na mesma frota de automóveis, esta pode ser considerada uma boa abordagem devido à ocorrência de *overfitting*. Estes treinos apresentam uma maior probabilidade de sucesso, porque a inferência ocorre muitas vezes em amostras de um dado carro já conhecido pelo método no treino. Para esta divisão usaram-se percentagens de 70% para o treino, 10% para a validação e 20% para o teste, estas são percentagens comummente atribuídas na avaliação destes algoritmos.
Na divisão Challenge, os conjuntos de treino, validação e teste foram preenchidos com carros isolados do *dataset*. O *dataset in-car* apresenta um total de 78 carros, sendo 8 deles carros novos (sem danos). Neste sentido, foram atribuídos 53 carros (70% do *dataset* total) ao conjunto de treino, sendo 5 dos carros novos (sem dano) e os restante 48 carros danificados. O conjunto de validação apresenta 8 carros no total (10% do *dataset* total), sendo 1 carro novo. O conjunto de teste contém um total de 17 carros (20% do *dataset* total), sendo 3 deles carros novos. A abordagem *Challenge* é a mais generalista, pois os métodos efectuam a sua inferência num conjunto de teste que contém carros nunca antes vistos pelo método, reduzindo assim a ocorrência de *overfitting*.

De forma a avaliar os modelos em abordagens de Challenge e Industrial, o *dataset in-car* foi distribuído de acordo com a Figura 71, esta é representada a distribuição percentual das classes contidas no *dataset in-car* para as abordagens Challenge e Industrial, respetivamente. É importante efectuar uma análise dos gráficos para perceber a distribuição percentual das classes contida nos conjuntos de treino, validação e teste das divisões do *dataset in-car*, estes conjuntos serão usados para os treinos na avaliação dos modelos de deteção de danos pré-seleccionados no capítulo 4.2.

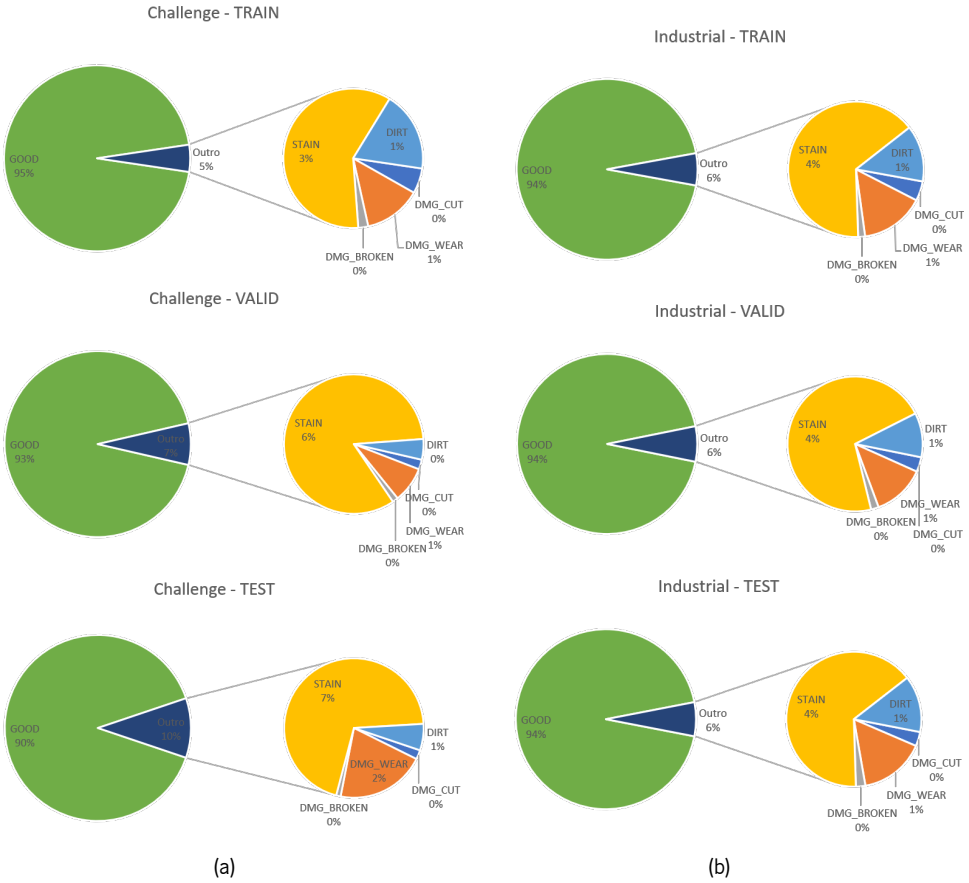


Figura 71: Gráficos circulares para apresentação da divisão percentual das classes encontradas no *dataset in-car*. (a) Conjuntos de divisão Challenge; (b) Conjuntos de divisão Industrial;

No capítulo 4.2 a rede que obteve a melhor performance, (i.e. OEV2), foi treinada para classificação das classes de dano diferenciadas e para a classificação de uma classe global que contém todas as classes de dano fundidas (i.e. OEV2.1). OEV2.1 apresentou uma exatidão superior em Challenge, comparativamente com OEV2, indicando que o método em fusão de classes poderá apresentar melhor capacidade de generalização. Dito isto, os métodos propostos serão treinados segundo duas abordagens: All Classes e Fused (Tabela 16). Na abordagem All Classes os métodos serão treinados para classificar individualmente as 5 classes presentes no nosso *dataset in-car*: CUT, WEAR, BROKEN, STAIN, DIRT. Na abordagem Fused as classes de dano (CUT,WEAR e BROKEN) são fundidas numa única denominada DAMAGE, mantendo-se as classes STAIN e DIRT. A fusão de classes é realizada a partir da Função Json2SEG_fusion para os métodos de *Semantic Segmentation*.

Os modelos DeepLabV3+ e U-Net são treinados segundo duas configurações de entrada de rede distintas:

Tabela 16: Definição de dois tipos de ensaios, em All Classes serão discriminadas todas as classes contidas no *dataset in-car*, em Fused as classes de dano: CUT, WEAR, BROKEN são fundidas numa só classe denominada DAMAGE.

	CLASSES
All Classes	CUT; WEAR; BROKEN; STAIN; DIRT
Fused	DAMAGE; STAIN; DIRT

Full e Tiled (Tabela 17 e Figura 72), de modo a interpretar qual é a configuração de entrada mais apropriada a cada um, sendo estas:

- **IEV1:** Avaliação do modelo DeepLabV3+ com uma resolução de entrada de rede de 1080x1920 para cada imagem;
- **IEV2:** Avaliação do modelo DeepLabV3+ para cada imagem de resolução de 1200x1200, cada imagem é decomposta em 4 imagens Tiled de 600x600, estas imagens Tiled são as entrada da rede do modelo.
- **IEV3:** Avaliação do modelo U-Net com uma resolução de entrada de rede de 1080x1920 para cada imagem;
- **IEV4:** Avaliação do modelo U-Net para cada imagem de resolução de 1200x1200, cada imagem é decomposta em 4 imagens Tiled de 600x600, estas imagens Tiled são as entrada da rede do modelo.

Tabela 17: Avaliação inicial das redes DeepLabV3+ e U-Net no *dataset in-car*, comparando e testando diferentes configurações de entrada de rede.

	Modelo	**Resolução de entrada**	**Depth**
IEV1	DeepLabV3+	Full at 1080x1920x3	-
IEV2	DeepLabV3+	Tiled at 600x600x3	-
IEV3	U-Net	Full at 1080x1920x3	3
IEV4	U-Net	Tiled at 600x600x3	3

Figura 72: Avaliações Full e Tiled. A imagem à esquerda mostra a rede do modelo a ser alimentada com a imagem completa do *dataset in-car*. A imagem à direita mostra a imagem do *dataset in-car* a ser divida em 4 partes, para entrar sequencialmente na rede do modelo.

Os ensaios da Tabela 17 foram realizados recorrendo a uma configuração similar: 100 epochs, otimizador ADAM, 0.001 de learning rate, learning rate drop factor de 30% a cada 10 epochs. O valor da batch size usado nos ensaios é de 2 e 4 IEV1/3 e IEV2/4, respetivamente. Os resultados dos ensaios propostos na Tabela 17, são apresentados nas Tabelas 18 e 19 para a abordagem Challenge e Industrial, respetivamente.

Tabela 18: Resultados dos ensaios do teste das diferentes configurações de entrada para os modelos DeepLabV3+ e U-Net na avaliação Challenge. A performance de cada ensaio é dada pela métrica Exatidão Média (%), o valor da Exatidão obtida em cada classe também é apresentado.

			mAC	GOOD	CUT	WEAR	BROKEN	STAIN	DIRT
Challenge	**All Classes**	**IEV1**	16.67%	100%	0%	0%	0%	0%	0%
		IEV2	**17.56%**	**89.40%**	**2.30%**	**1.54%**	**0.56%**	**9.88%**	**1.66%**
		IEV3	16.67%	100%	0%	0%	0%	0%	0%
		IEV4	**18,26%**	**82.08%**	**5.46%**	**6.56%**	**2.16%**	**11.93%**	**1.36%**
			mAC	**GOOD**	**DAMAGE**			**STAIN**	**DIRT**
	Fused	**IEV1**	25.0%	100%	0%			0%	0%
		IEV2	**26.64%**	**90.34%**	**7.75%**			**5.83%**	**2.64%**
		IEV3	25.0%	100%	0%			0%	0%
		IEV4	**27.55%**	**80.71%**	**12.8%**			**14.75%**	**3%**

Tabela 19: Resultados dos ensaios do teste das diferentes configurações de entrada para os modelos DeepLabV3+ e U-Net na avaliação Industrial. A performance de cada ensaio é dada pela métrica Exatidão Média (%), o valor da Exatidão obtida em cada classe também é apresentado.

			mAC	GOOD	CUT	WEAR	BROKEN	STAIN	DIRT
Industrial	**All Classes**	**IEV1**	16.67%	100%	0%	0%	0%	0%	0%
		IEV2	**17.32%**	**88.28%**	**2.75%**	**2.03%**	**0.92%**	**8.28%**	**1.66%**
		IEV3	18,21%	96.85%	0%	0%	0%	12.38%	0%
		IEV4	**18.82%**	**79.02%**	**6.43%**	**6.87%**	**3.12%**	**15.26%**	**2.21%**
			mAC	**GOOD**	**DAMAGE**			**STAIN**	**DIRT**
	Fused	**IEV1**	**35.96%**	**89.62%**	**6.95%**			**21.41%**	**25.87%**
		IEV2	26,10%	91.88%	3.99%			6.48%	2.05%
		IEV3	29.77%	94.20%	0%			24.61%	0%
		IEV4	**27,92%**	**81.14%**	**10.86%**			**15.88%**	**2.78%**

Após a avaliação dos resultados apresentados nas Tabelas 18 e 19, conclui-se que as duas configurações de entrada de rede seleccionadas para cada modelo são, i.e. IEV1 e IEV4. Para a melhor configuração (i.e. Fused em Industrial), foram realizados mais estudos de ablação e hiperparâmetros para ambos os modelos: DeepLabV3+ e U-Net. Este estudo é apresentado a seguir na Secção 5.2.

5.2 Optimização de algoritmos

Após a seleção da melhor configuração para o modelo DeepLabV3+, IEV1, foram efectuados novos ensaios ao nível de resolução de entrada e batch (Tabela 20). Os resultados podem ser vistos na Tabela 21.
No modelo U-Net a configuração de entrada de rede que apresentou a melhor performance foi a Tiled, seguindo esta abordagem, novos valores de resolução de entrada foram testados, combinados com novos valores de depth encoder e diferentes valores de batch (Tabela 22), os resultados são apresentados iterativamente na Tabela 23. Todos os ensaios partem de uma imagem de 1024x1024x3, originando 4 Tiles de 512x512x3 e 16 de 256x256x3.
A melhor performance foi encontrada no ensaio IEV1.4 que consiste no modelo DeepLabV3+ com resolução

Tabela 20: Estudo de ablação e hiperparâmetros para DeepLabV3+ (IEV1). As avaliações com melhor performance são apresentadas a negrito e a cor verde iterativamente quando esses valores foram encontrados.

	Resolução de entrada	Batch
IEV1.1	Full at 1024x1024x3	2
IEV1.2	**Full at 512x512x3**	2
IEV1.3	Full at 512x512x3	4
IEV1.4	Full at 512x512x3	**8**
IEV1.5	Full at 512x512x3	16

Tabela 21: Resultados do estudo de ablação e hiperparâmetros para o modelo DeepLabV3+. A linha a negrito e cor verde é a que apresenta melhor performance dada pela métrica Exatidão Média (%), o valor da Exatidão obtida em cada classe também é apresentado.

	mAC	GOOD	DAMAGE	STAIN	DIRT
IEV1.1	34.03%	92.96%	20.43%	14.62%	8.11%
IEV1.2	45.07%	82.62%	29.65%	45.53%	22.47%
IEV1.3	60.79%	77.82%	47.06%	58.81%	59.47%
IEV1.4	**67.60%**	**77.17%**	**59.60%**	**66.81%**	**68.82%**
IEV1.5	67.52%	75.05%	62.86%	64.67%	67.54%

Tabela 22: Estudo de ablação e hiperparâmetros para o modelo U-Net (IEV4). As avaliações que apresentam as melhores performances são apresentadas a negrito e a cor verde iterativamente à medida que são encontradas.

	Resolução de entrada	Depth	Batch
IEV4.1	**Tiled at 512x512x3**	**3**	4
IEV4.2	Tiled at 256x256x3	3	4
IEV4.3	Tiled at 512x512x3	4	4
IEV4.4	Tiled at 512x512x3	3	**8**

Tabela 23: Resultados do estudo de ablação e hiperparâmetros para o modelo U-Net. A linha a negrito e cor verde é a que apresenta melhor performance dada pela métrica Exatidão Média (%), o valor da Exatidão obtida em cada classe também é apresentado.

	mAC	GOOD	DAMAGE	STAIN	DIRT
IEV4.1	27.97%	80.71%	10.86%	15.88%	2.78%
IEV4.2	25.18%	96.34%	2.42%	1.56%	0.38%
IEV4.3	27.80%	80.56%	12.8%	15.17%	2.67%
IEV4.4	**28.37%**	**71.50%**	**19.24%**	**18.43%**	**4.15%**

de entrada de rede de 512x512x3 Full image com batch de 8, a Exatidão Média obtida foi de 67.60% para a avaliação com classes Fused (DAMAGE, STAIN e DIRT) com divisão de *dataset* Industrial. O melhor ensaio encontrado (i.e. IEV1.4) DeepLabV3+ com resnet18, foi então replicado para as divisões Challenge e Industrial segundo avaliação de classes: All Classes e Fused, os resultados são apresentados nas Tabelas 24 e 25.

Nas Figuras 73 e 74 são apresentados resultados das inferências do método DeepLabV3+ (i.e. IEV1.4) com divisão Challenge no ensaio de classes: All Classes (Exatidão Média de 40.95%) e Fused (Exatidão Média de 57.26%), respetivamente.

Nas Figuras 75 e 76 são apresentados resultados das inferências do método DeepLabV3+ (i.e. IEV1.4) com

Tabela 24: Resultados do modelo DeepLabV3+ na avaliação Challenge. A performance de cada ensaio é dada pela métrica Exatidão Média (%), o valor da Exatidão obtida em cada classe também é apresentado.

			mAC	GOOD	CUT	WEAR	BROKEN	STAIN	DIRT
Challenge	**All Classes**	**IEV1.4**	40.95%	63.74%	25.31%	33.70%	22.41%	50.54%	50.03%
			mAC	**GOOD**	**DAMAGE**			**STAIN**	**DIRT**
	Fused	**IEV1.4**	57,26%	70.94%	39.79%			60.17%	58.16%

Tabela 25: Resultados do modelo DeepLabV3+ na avaliação Industrial. A performance de cada ensaio é dada pela métrica Exatidão Média (%), o valor da Exatidão obtida em cada classe também é apresentado.

			mAC	GOOD	CUT	WEAR	BROKEN	STAIN	DIRT
Industrial	**All Classes**	**IEV1.4**	53.94%	69.67%	47.44%	47.62%	35.77%	61.97%	60.18%
			mAC	**GOOD**	**CUT**	**DAMAGE**			**DIRT**
	Fused	**IEV1.4**	67.60%	77.17%	58.60%	65.81%			68.82%

divisão Industrial no ensaio de classes: All Classes (Exatidão Média de 53.94%) e Fused (Exatidão Média de 67.60%), respetivamente.

5.3 Discussão e conclusão

Nesta secção é proposto o uso de métodos de *Semantic Segmentation* para a deteção de danos e sujidade no interior dos carros. Para o estudo foi criado um *dataset* com imagens do interior de 78 carros (total 1861 imagens), sob diferentes perspectivas. Seguidamente, foram treinados dois métodos de Semantic Segmentation: DeepLabV3+ e U-Net no *dataset in-car*. Inicialmente, as duas redes foram avaliadas ao nível de configuração de entrada (i.e. Full Image e Tiled) e de *dataset* (i.e. Industrial e Challenge). Os resultados (Tabelas 18 e 19) mostraram que o DeepLabV3+ obtém maior precisão ao usar a abordagem de entrada de rede Full image, IEV1, já a U-Net teve um desempenho superior com a abordagem com Tiles, IEV4. A partir destas duas metodologias, foi realizado um estudo de ablação e hiperparâmetros (Tabelas 20 e 22) para cada uma, com o intuito de obter a melhor Exatidão possível. Os resultados mostraram (Tabela 23) que U-Net alcançou a maior Exatidão em IEV4.4, com entrada Tiled em 512x512x3, encoder depth de 3 e batch 8, atingindo uma Exatidão de 28,37%, 71,50%, 19,24%, 18,43% e 4,15% de média, GOOD, DAMAGE, STAIN e DIRT, respectivamente. Além disso, DeepLabV3+ superou U-Net consideravelmente em IEV1.4 (Tabela 21), com uma entrada de rede Full image de 512x512x3 e batch 8, atingindo uma Exatidão de 67,60%, 77,17%, 59,60%, 66,81% e 68,82% de média, GOOD, DAMAGE, STAIN e DIRT, respectivamente. Em relação ao DeepLabV3+, após uma breve comparação entre as opções de resoluções de entrada, concluiu-se que a abordagem que obteve a melhor Exatidão, foi 512x512x3 em vez de 1024x1024x3, este resultado é influenciado pela perda de informação do pixel de classe com a redução de resolução, auxiliando deste modo a convergência do treino. Embora a inferência de classe seja geralmente boa, às vezes há uma troca entre as classes danificadas (Figuras 75 e 76) quando a distinção entre elas não é tão aparente, na realidade, mesmo a distinção visual para humanos é difícil, pois a aparência de algumas classes pode ser muito semelhante dependendo do tipo de tecido. No caso do U-Net, apesar do estudo de ablação e hiperparâmetros, verificou-se

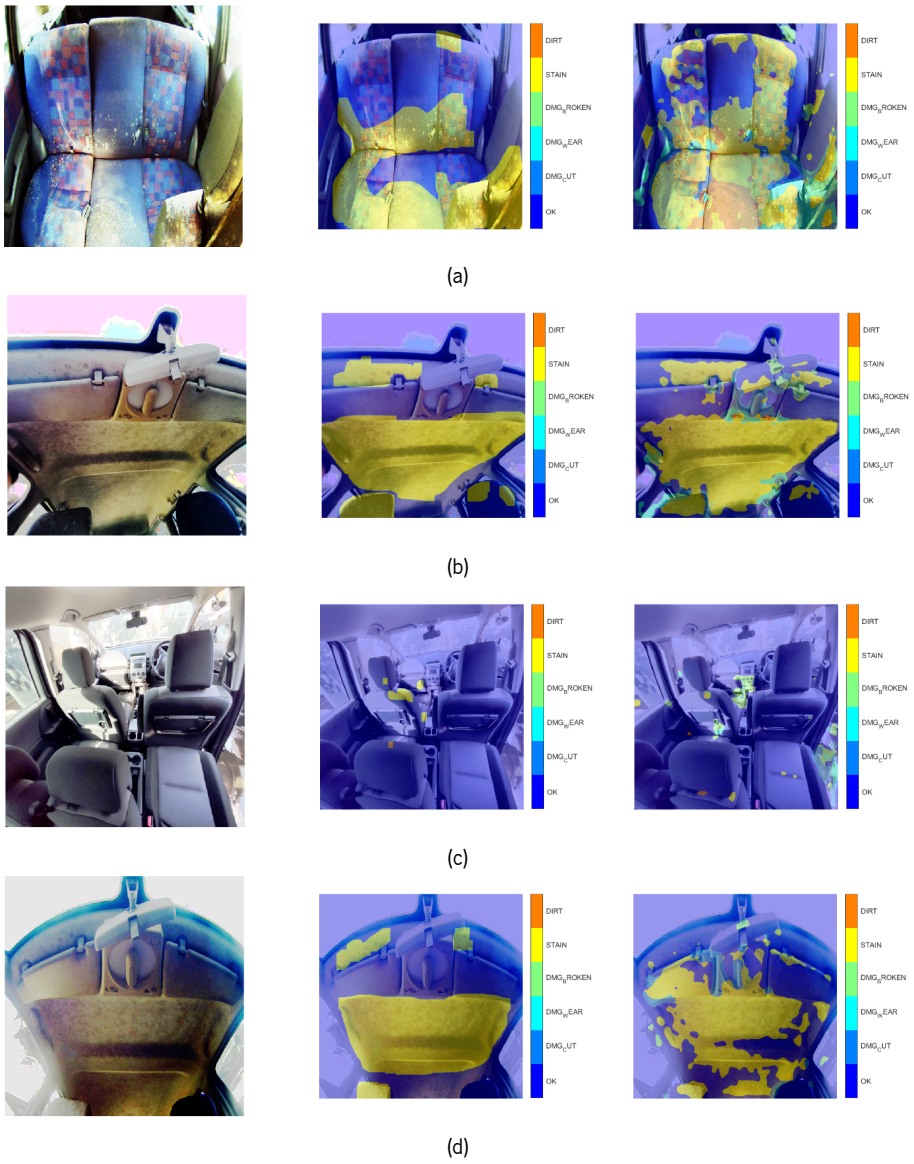


Figura 73: Exemplos da inferência do método DeepLabV3+ com divisão Challenge no ensaio All CLasses.
(a) Vista para o banco traseiro de um carro do conjunto de teste, o método classifica como dano as zonas não saudáveis do material, verificamos que onde existe a classe STAIN o método classificou maioritariamente como STAIN, excepto a parte central do banco que classificou como DIRT, e umas zonas laterais como WEAR. O que se verifica é que como o banco do carro é forrado com tecido com padrão isto pode complicar e confundir a deteção do método. (b) Vista para o teto de um carro do conjunto do teste, o método classifica toda a zona danificada atribuindo a classe correta STAIN, exceto nas periferias que confunde com a classe WEAR (cor azul). (c) Vista geral do fundo do carro, o método não acerta totalmente a classificação dos danos, mas classifica quase na totalidade a zona boa do carro, o que também é muito positivo.(d) Vista para o teto de um carro, a classificação não perfeita, conseguindo acertar apenas algumas partes da classe DIRT presente.

que neste tipo de abordagem e estudo, esta rede apresenta uma exatidão muito menor comparando com o DeepLabV3+. Em suma, após a otimização foi possível chegar ao ensaio IEV1.4, este foi replicado segundo as divisões Industrial e Challenge com All Classes e Fused (Tabelas 24 e 25), a partir desta configuração foi possível obter resultados consideravelmente superiores em todas as configurações.

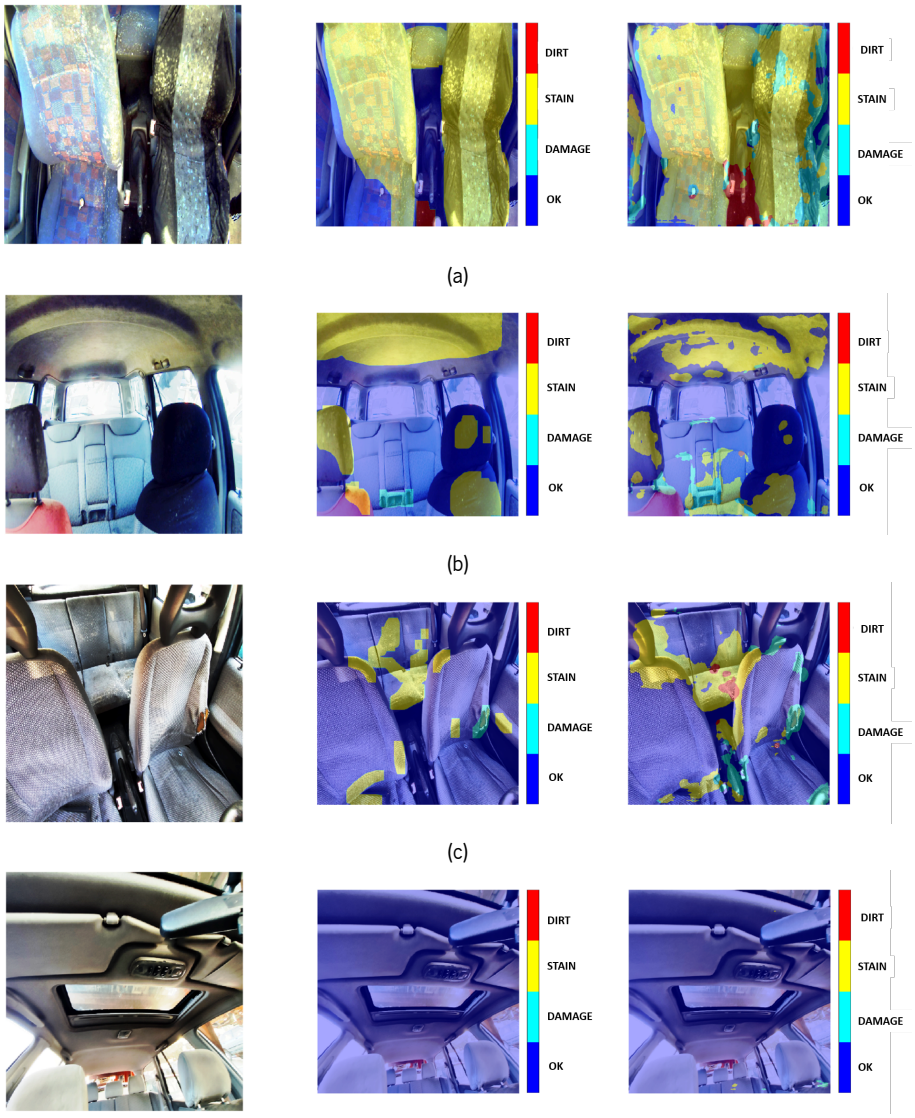


Figura 74: Exemplos da inferência do método DeepLabV3+ (i.e. IEV1.4) com divisão Challenge no ensaio Fused. (a) Vista sobre os bancos da frente, observa-se que quase toda a área danificada foi classificada como material danificado, mas certas zonas atribuídas como STAIN pelo *labeling* foram classificadas pela classe DAMAGE. (b) Vista sobre os bancos traseiros, classificação na generalidade boa. (c) Vista sobre os bancos da frente de um carro do conjunto de teste, observa-se uma classificação bastante boa, com atribuição das classes de dano praticamente em coerência com as do *labeling*. (d) Vista para o teto de um carro do conjunto de teste, material que não apresenta danos é classificado corretamente pelo método segundo a classe OK.

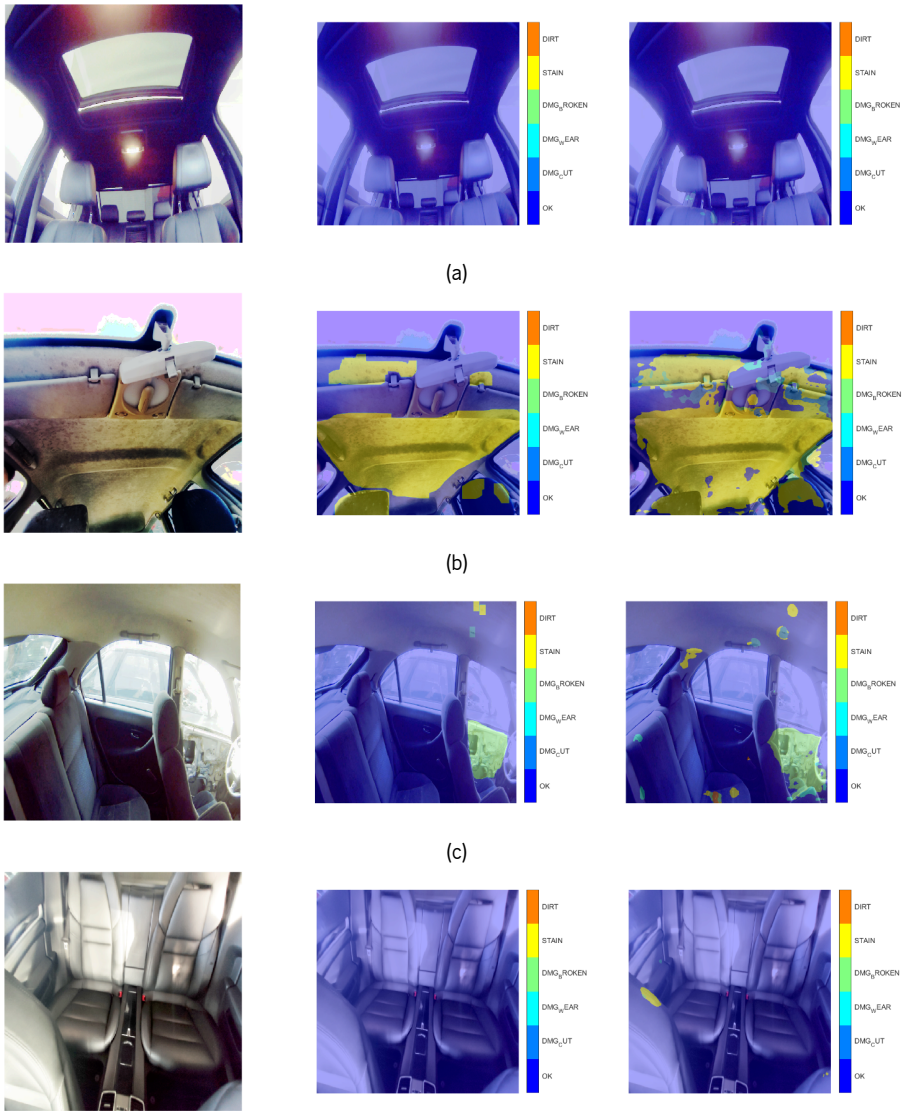


Figura 75: Exemplos da inferência do método DeepLabV3+ (i.e. IEV1.4) com divisão Industrial no ensaio All Classes. (a) Vista sobre o teto de um carro do conjunto de teste, a imagem não apresenta danos, e é corretamente classificada segundo a classe OK. (b) Vista sobre o teto de um carro do conjunto de teste, praticamente toda a área classificada como STAIN é corretamente classificada excepto pequenos que o método classificou como WEAR (cor azul), talvez por influencia da luz incidente. (c) Vista lateral esquerda do banco traseiro, verifica-se uma inferência muito boa. (d) Vista sobre o banco traseiro de um carro semi-novo do conjunto de teste, quase toda a imagem é classificada corretamente com a classe OK, excepto uma pequena área que o método classificou como WEAR, talvez por influência do reflexo da luz.

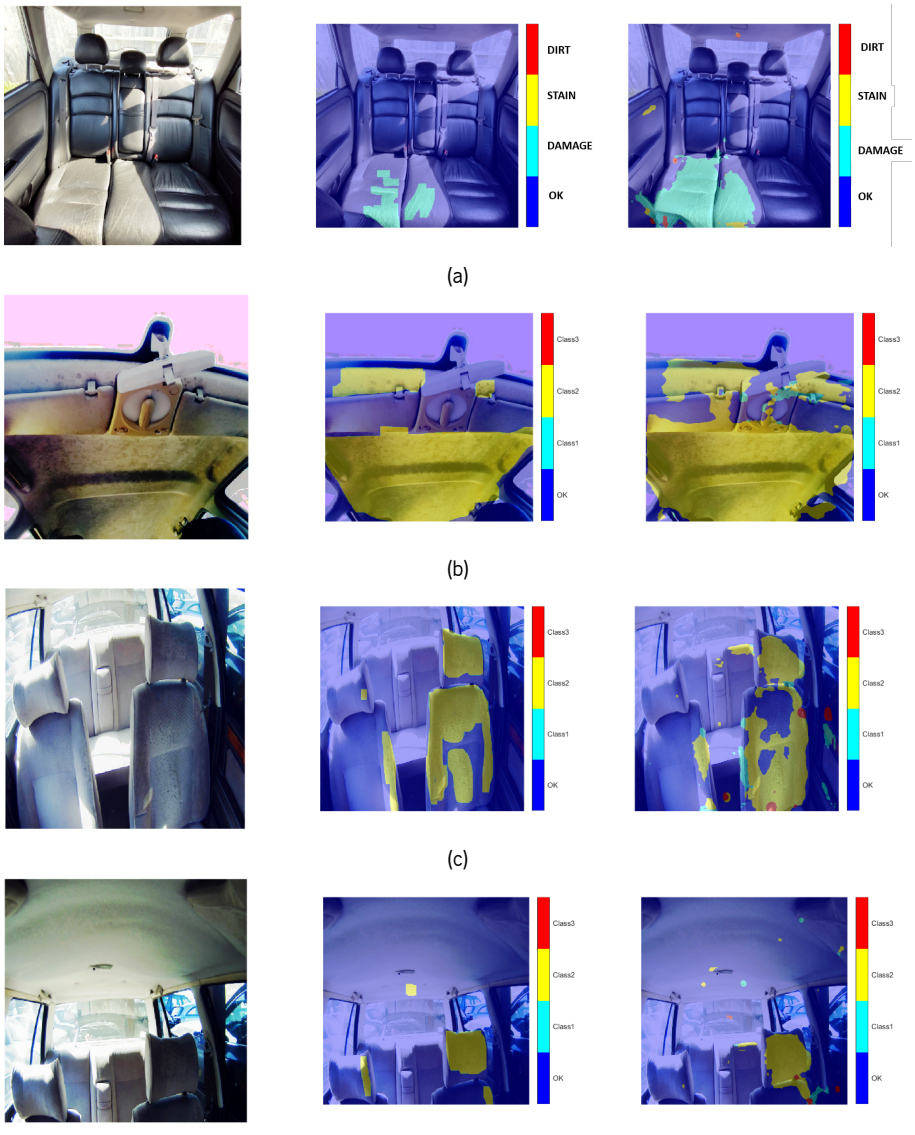


Figura 76: Exemplos da inferência do método DeepLabV3+ (i.e. IEV1.4) com divisão Industrial no ensaio Fused. (a) Vista sobre os bancos traseiros de um carro semi-novo do conjunto de teste, verifica-se uma atribuição correta da classe DAMAGE, mas sobre uma superfície maior que a atribuída ao nível do *labeling*, este efeito deve-se também ao modo como o método efectua a classificação, quando zonas da mesma classe estão muito juntas, o método apresenta essas zonas todas numa só. (b) Vista sobre o teto de um carro do conjunto de teste, inferência boa. (c) Vista sobre os bancos da frente de um carro do conjunto de teste, inferência boa. (d) Vista sobre todo o carro a partir da frente, inferência razoável.

6

CONCLUSÃO E TRABALHO FUTURO

Neste capítulo resumem-se as conclusões obtidas durante o tempo dedicado à elaboração desta dissertação e definem-se algumas linhas de trabalho e sugestões para trabalho futuro.
O desenvolvimento desta dissertação teve como objetivo a criação de um algoritmo capaz de classificar danos em tecido no interior de um carro, para além da sua classificação também era importante saber a sua distribuição espacial.

6.1 Conclusão

Nesta dissertação propôs-se a avaliação de algoritmos capazes de detetar danos em tecido no interior de um veículo. O grande desafio é conseguir efectuar uma boa inferência, onde as condições de acessibilidade e luminosidade não são as melhores.
Inicialmente foi necessário identificar potenciais algoritmos na deteção de danos em contexto automóvel ou têxtil. Os algoritmos escolhidos que se mostraram potenciais candidatos na resolução desta problemática foram métodos de *Object Detection* e de *Semantic Segmentation*.
Para o teste inicial destes métodos foi necessário gerar um *dataset* denominado *out-car* onde se recolheram imagens de amostras de tecido existente em interiores de carros. Posto isto, realizou-se *labeling* manual sobre este leque de imagens considerando as seguintes classes: Corte, Furo, Desgaste e Queimadura. Nos métodos de *Object Detection* foi avaliado o YOLOV2, relativamente a *Semantic Segmentation*, foram avaliados dois métodos, U-Net e DeepLabV3+, onde o último foi avaliado com diferentes *backbones*. Os resultados finais mostraram que os métodos de *Semantic Segmentation* apresentam uma boa oportunidade de serem aplicados neste contexto de interior de veiculo.
Uma vez realizada a pré- seleção dos melhores algoritmos na deteção de danos em tecido procedeu-se à avaliação destes em contexto de interior de veiculo, para isso foi criado um *dataset in-car* onde foram capturadas e reunidas imagens do interior de carros velhos e semi-novos. Depois de reunidas as imagens procedeu-se mais uma vez ao *labeling* manual considerando as seguintes classes: CUT, WEAR, BROKEN, STAIN e DIRT. O método de *Object Detection* foi descartado na avaliação de danos no interior dos veículos por não apresentar boa capacidade de convergência e generalização nesta problemática. Por outro lado os métodos U-Net e DeepLabV3+ foram os que obtiveram melhores resultados, com o método DeepLabV3+ a obter uma inferência bastante satisfatória, apresentando uma Exatidão Média de 67.60%, para uma configuração de 512x512x3 de input de rede, IEV1.4, o que superou os objetivos iniciais expectáveis que eram de obter um resultado entre 40% e 50%.

6.2 Trabalho futuro

Como trabalho futuro, o objetivo é expandir o *dataset in-car*, tentando adicionar mais amostras e mais variabilidade ao nível dos carros e classes encontradas neste contexto. Neste poderão ser adicionadas novas labels, nomeadamente uma avaliação humana do nível de qualidade do interior do veiculo. Permitindo assim a avaliação de métodos que consigam não só estimar a presença de dano, sujidade e manchas, bem como classificar a qualidade do serviço de forma qualitativa (i.e. baseado na subjectividade humana).

Anexos

A

GESTÃO DE DATASETS

Function SplitDatasetV1(train, valid, featurepath, labelpath, outputpath)

train: Percentagem atribuída ao conjunto de treino;
valid: Percentagem atribuída ao conjunto de validação;
featurepath: Caminho absoluto das features;
labelpath: Caminho absoluto das labels;
outputpath: Caminho absoluto pretendido onde as divisões relativas ao treino e validação serão guardadas;
Exemplo:
SplitDatasetV1(0.6,0.2,'C:\Users\SandraDixe\Desktop\Tese\DatasetOutCar\Features','C:\Users\SandraDixe\Desktop\Tese\DatasetOutCar\Labels','C:\Users\SandraDixe\Desktop\Tese\DatasetOutCar\SplitV1')

Function SplitDatasetV2(train, valid, test, featurepath, labelpath, outputpath)

train: Amostras atribuídas ao conjunto de treino;
valid: Amostras atribuídas ao conjunto de validação;
test: Amostras atribuídas ao conjunto de validação;
featurepath: Caminho absoluto das features;
labelpath: Caminho absoluto das labels;
outputpath: Caminho absoluto pretendido onde as divisões relativas ao treino e validação serão guardadas;
Exemplo: SplitDatasetV1({'S1','S7','S8','S10'},{'S1'},{'S1','S7'},'C:\Users\SandraDixe\Desktop\Tese\DatasetOutCar\Features','C:\Users\SandraDixe\Desktop\Tese\DatasetOutCar\Labels','C:\Users\SandraDixe\Desktop\Tese\DatasetOutCar\SplitV2')

Function TrainSEG(datasetinputpath, datasetoutputpath, Sfactor)

datasetinputpath: Caminho absoluto do dataset que contém as features e labels;
datasetoutputpath: Caminho absoluto onde será guardado o dataset (features e labels) já redimensionadas;
Sfactor: Factor de redimensionamento aplicado às imagens do dataset;
Exemplo: TrainSEG('C:\Users\SandraDixe\Desktop\Tese\DatasetOutCar\SplitV1','C:\Users\SandraDixe\Desktop\Tese\DatasetOutCar\SplitV1\Resize',[1200;1600])

Function FixRoot(jsonpath, imgrootpath, segrootpath)

jsonpath: Caminho absoluto do ficheiro JSON no novo computador;
imgrootpath: Caminho absoluto das IMAGES do DATASET V3.0.1;
segrootpath: Caminho absoluto das SEGMENTATIONS do DATASET V3.0.1.;
Exemplo:
FixRoot('C:\Datasets\Dataset-incar\DATASETV3.0.1\dataset.json','C:\Datasets\Dataset-incar\DATASETV3.0.1\IMAGES','C:\Datasets\Dataset-incar\DATASETV3.0.1\SEGMENTATIONS')

Function RawPath2Json(datasetpath, outputpath)

datasetpath: Caminho que aponta para a pasta com todas as SEGMENTATIONS e IMAGES dos carros;
outputpath: Caminho onde o ficheiro JSON gerado será criado;
Exemplo: RawPath2Json('C:\Datasets\Dataset-incar\DATASETV2.0','C:\Datasets\Dataset-incar\DATASETV2.0)

Function ImagesCounter(file, path)

file: Local do ficheiro JSON com as IMAGES e SEGMENTATIONS;
path: Caminho onde o ficheiro Excel será criado;
Exemplo: ImagesCounter('C:\Datasets\Dataset-incar\DATASETV3.0.1\dataset.json','C:\Datasets\Dataset-incar\DATASETV3.0.1\reports)

Function FilterDataset(inputpath, filteredjsonpath, outputpath, Brand, Model, Type, Ceiling, Plastics, Seats, View)

inputpath: Localização do ficheiro JSON com IMAGES e SEGMENTATIONS;
filteredjsonpath: Localização do ficheiro JSON com a filtragem anterior (ou seja, executar uma condição OR), este parâmetro é opcional;
outputpath: Localização onde o ficheiro JSON filtrado será criado;
Brand: Cadeia de filtragem por marca de carro;
Model: Cadeia de filtragem por modelo de carro;
Type: Cadeia de filtragem por tipo de carro;
Ceiling: Cadeia de filtragem por cor do teto do carro;
Plastics: Cadeia de filtragem por cor do plástico interior de carro;
Seats: Cadeia de filtragem por cores de assentos de carro;
View: Cadeia de filtragem por posições da câmara (P1 a P9);
Exemplo: FilterDataset('C:\Datasets\Dataset-incar\DATASETV3.0.1\dataset.json','C:\train.json','renault','clio','','','black','','P1','P2','P3','P4','P7','P8','P9')

Function RawPathAndJsonFuseSegmentation(file, segpath, outputpath)

file: Localização do ficheiro JSON com IMAGES e SEGMENTATIONS;
segpath: Caminho do dataset que contém as labels da Segmentação.;
outputpath: Caminho onde o ficheiro JSON será actualizado (ou seja, resultado da fusão da informação da estrutura do dataset com informações de segmentação);
Exemplo:
RawPathAndJsonFuseSegmentation('C:\Datasets\Dataset-incar\DatasetV2.0\dataset.json','C:\Datasets\Datasetin-car\DatasetV2.0.1','C:\Datasets\Datasetin-car\DatasetV3.0.1')

B

MÉTODOS

Function Json2OBD(jsonpath, obdpath, sfactor))

jsonpath: Caminho que aponta para um JSON completo ou filtrado. O ficheiro JSON requer que os caminhos das IMAGES E SEGMENTATIONS do dataset estejam devidamente actualizados, caso contrário, é necessário executar o algoritmo FixRoot.m;

obdpath: Caminho para onde o ficheiro .mat resultante desta função será alvo. Este ficheiro armazena o caminho das imagens e as respectivas bounding boxes para cada classe. Para o cálculo das bounding boxes, recorre ao algoritmo 2;

sfactor: Fator opcional de redimensionamento para as IMAGES e SEGMENTATIONS;

Exemplo: Json2OBD('C:\Articles\Evaluations\jsons\train.json','C:\Articles\Evaluations\OBD\EV1\train.mat',[1080;1920])

Function Json2OBD_fusion(jsonpath, obdpath, sfactor, classesfusion))

jsonpath: Caminho que aponta para um JSON completo ou filtrado. O ficheiro JSON requer que os caminhos das IMAGES E SEGMENTATIONS do dataset estejam devidamente actualizados, caso contrário, é necessário executar o algoritmo FixRoot.m;

obdpath: Caminho para onde o ficheiro .mat resultante desta função será alvo. Este ficheiro armazena o caminho das imagens e as respectivas bounding boxes para cada classe. Para o cálculo das bounding boxes, recorre ao algoritmo 2;

sfactor: Fator opcional de redimensionamento para as IMAGES e SEGMENTATIONS.;

classesfusion: Classes seleccionadas para fusão, as classes são apresentadas pelo seu PixelID, todas as classes separadas por virgula são fundidas numa só. As classes separadas por ponto e vírgula são divididas. **Exemplo:** Json2OBD_fusion('C:\Articles\Evaluations\jsons\train.json','C:\Articles\Evaluations\OBD\EV1\train.mat',[1080;1920],['1,2,3;4,5'])

Function Json2SEG(jsonpath, segpath, sfactor)

jsonpath: Caminho do JSON completo ou filtrado. O ficheiro JSON requer que os caminhos das IMAGES e SEGMENTATIONS do dataset estejam devidamente actualizados, caso contrário, é necessário executar o algoritmo FixRoot.m.;

segpath: Caminho que aponta para a pasta criada que armazenará: uma subpasta das SEGMENTATIONS, uma subpasta das IMAGES e um arquivo .mat com as classes e o ClassPixelID;

sfactor: Fator opcional de redimensionamento para as IMAGES e SEGMENTATIONS;

Exemplo: Json2SEG('C:\Articles\Evaluations\jsons\train.json','C:\Articles\Evaluations\SEG\EV1\train',[1080;1920]

Function Json2SEG_fusion(jsonpath, segpath, sfactor,classesfusion)

jsonpath: Caminho do JSON completo ou filtrado. O ficheiro JSON requer que os caminhos das IMAGES e SEGMENTATIONS do dataset estejam devidamente actualizados, caso contrário, é necessário executar o algoritmo FixRoot.m.;

segpath: Caminho que aponta para a pasta criada que armazenará: uma subpasta das SEGMENTATIONS, uma subpasta das IMAGES e um arquivo .mat com as classes e o ClassPixelID;

sfactor: Fator opcional de redimensionamento para as IMAGES e SEGMENTATIONS;

classesfusion: Classes seleccionadas para fusão, as classes são apresentadas pelo seu PixelID, todas as classes separadas por virgula são fundidas numa só. As classes separadas por ponto e vírgula são divididas. **Exemplo:** Json2SEG('C:\Articles\Evaluations\jsons\train.json','C:\Articles\Evaluations\SEG\EV1\train',[1080;1920],['1,2,3;4,5']

Algorithm 1: Algoritmo transversal aos modelos de Semantic Segmentation e Object Detection.

if *istrain* **then**

(1) Load dos datasets: Aceder às *features* e *labels*, para isto, são criados dois *datastores* do MATLAB que servem para guardar a informação destes dados; definição das classes; definição dos pixeis IDs correspondentes a cada classe.

traindata = datasetpath\train;

validdata = datasetpath\valid;

if *isresume* **then**

net = load(modeloantigo);

else

(2) Criação do modelo: Criação de modelo segmentador com ablação (i.e. DeepLabV3+ ou U-Net, com alterações de backbone, resolução de entrada, depth de encoding/decoding, número de classes e pesos atribuídos às classes no cálculo da loss).

if *U-Net DeepLabV3+* **then**

% obter frequência de cada classe no dataset de treino

$\text{frequenciaclasse}_c = \frac{1}{totalpixeisclasse_c}$

$\text{pesoclasse}_c = \frac{frequenciaclasse_c}{\left(\frac{\sum_{c=1}^{numeroclasses} frequenciaclasse_c}{numeroclasses}\right)}$

if *U-Net* **then**

net = unet(resolução, depth, numeroclasses, pesoclasses);

if *DeepLabV3+* **then**

net = deeplabv3(resolução, backbone, numeroclasses, pesoclasses);

else if *YoloV2* **then**

% calcular anchor boxes dos dados de treino (i.e. Figura 65)

anchors = anchorboxes(traindata);

net = yolov2(resolução, backbone, numeroclasses);

(3) Definição dos parâmetros de treino: Seleccionar as opções de treino, Figura 68;

trainoptions = Figura 68;

(4) Treino e validação do modelo;

net = train(net,trainoptions,traindata,validdata);

(5) Teste do modelo: Avaliação da rede treinada.

if *istest* **then**

testdata = datasetpath\test;

deteção = estimate(testdata(features),net);

metricas = evaluate(deteção,testdata(labels));

Algorithm 2: Função auxiliar que calcula as *bounding boxes* através das *labels* (máscaras criadas no *Ground Truth Labeler*.)

For c=1 : numeroclasses
    mask = label(c); % mask é a binarização de onde a classe, c, se encontra na label;
    mask = imdilate(mask); % função dilata a mask e corrige alguma falha de *labeling* caso exista;
    bb = regionprops(maskdilated, 'BoundingBox'); % esta função, do Matlab, determina a posição e coordenadas de todas as *bounding boxes* que uma mask apresenta para a classe c;
    row(c) = bb
retorna *row* com a informação de todas as *bounding boxes* da respectiva imagem;

Function Object_Detection(datasetpath, modelpath, lr, solver, epochs, batch, nanchors, isanchors, isplotanchors, istrain, isresume, istest)

Datasetpath: Localização do *dataset* para carregar os ficheiros *train.mat*, *valid.mat* e *test.mat*. Esses arquivos foram gerados anteriormente pelas Funções Json2OBD e Json2OBD_fusion;

Moldelpath: Caminho para onde modelos e as métricas resultantes serão salvas;

Lr: Learning rate é um hiperparâmetro que controla a rapidez com que a rede neuronal actualiza os conceitos que aprendeu. Um *Learning rate* desejável é aquele que, é baixo o suficiente para a rede convergir para algo útil, e alto o suficiente para ser treinada num período de tempo razoável;

Solver: Algoritmo de otimização;

Epochs: Número de vezes que o algoritmo de OD passa pelo conjunto de treino do *dataset*, ou seja, este número representa o número de vezes que o *dataset* é transmitido para a frente e para trás na rede;

Batch: É o número de amostras que são trabalhadas de todas as vezes ao mesmo tempo antes de actualizarem os parâmetros internos do modelo;

Nanchors: Estima o numero de *anchor boxes* definido para Nanchor, a partir do *dataset* de treino;

Isanchors: *Flag* para o cálculo de *anchor boxes*;

Isplotanchors: *Flag* para fazer desenhar gráfico que relaciona o número de *anchor boxes* com a métrica *mean* IOU;

Istrain: *Flag* para realização do treino;

Isresume: *Flag* para realização do resumo, se esta *flag* for *true* o modelo efectua o *load* do ficheiro correspondente a um treino anterior feito disponível no caminho *Modelpath*;

Istest: *Flag* para realização do teste;

Exemplo: Object_Detection('C:\Datasets\Datasetout-car\ObjectDetection','C:\Datasets\Datasetout-car\ObjectDetection\results\EV4\T1',0.01,'sgdm',100,2,15,true,false,true,true,false)

Function Semantic_Segmentation(datasetpath, modelpath, lr, solver, epochs, batch, istrain, isresume, istest)

Datasetpath: Localização do *dataset* para carregar as features e as labels. Estes arquivos foram gerados anteriormente pelas Funções Json2SEG e Json2SEG_fusion;

Moldelpath: Caminho para onde modelos e as métricas resultantes serão salvas;

Lr: Learning rate é um hiperparâmetro que controla a rapidez com que a rede neuronal actualiza os conceitos que aprendeu. Um *Learning rate* desejável é aquele que, é baixo o suficiente para a rede convergir para algo útil, e alto o suficiente para ser treinada num período de tempo razoável;

Solver: Algoritmo de otimização;

Epochs: Número de vezes que o modelo passa pelo conjunto de treino do *dataset*, ou seja, este número representa o número de vezes que o *dataset* é transmitido para a frente e para trás na rede;

Batch: É o número de amostras que são trabalhadas de todas as vezes ao mesmo tempo antes de actualizarem os parâmetros internos do modelo;

Istrain: *Flag* para realização do treino;

Isresume: *Flag* para realização do resumo, se esta *flag* for *true* o modelo efectua o *load* do ficheiro correspondente a um treino anterior feito disponível no caminho *Modelpath*;

Istest: *Flag* para realização do teste;

Exemplo: Semantic_Segmentation('C:\Datasets\Datasetout-car\SemanticSegmentation','C:\Datasets\Datasetout-car\SemanticSegmentation\results\EV4\T1',0.01,'sgdm',100,2,15,true,false,true,true,false)

7

BIBLIOGRAFIA

Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Boyd Cohen and Jan Kietzmann. Ride On! Mobility Business Models for the Sharing Economy. *Organization & Environment*, 27(3):279–296, sep 2014. ISSN 1086-0266. doi: 10.1177/1086026614546199. URL http://journals.sagepub.com/doi/10.1177/1086026614546199.

Abhipraya Kumar Dash. Single shot detection (ssd) algorithm, 2018. URL https://iq.opengenus.org/single-shot-detection-ssd-algorithm/. Último acesso em 2 de Junho de 2020.

Priya Dwivedi. Semantic segmentation — popular architectures, 2019. URL https://towardsdatascience.com/semantic-segmentation-popular-architectures-dff0a75f39d0. Último acesso em 15 de Junho de 2020.

Daniel J. Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, jul 2015. ISSN 09658564. doi: 10.1016/j.tra.2015.04.003.

Daniel J. Fagnant and Kara M. Kockelman. The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transportation Research Part C: Emerging Technologies*, 40:1–13, mar 2014. ISSN 0968090X. doi: 10.1016/j.trc.2013.12.001. URL https://www.sciencedirect.com/science/article/pii/S0968090X13002581.

Daniel J. Fagnant and Kara M. Kockelman. Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas. *Transportation*, 45(1):143–158, jan 2018. ISSN 15729435. doi: 10.1007/s11116-016-9729-z.

Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, Sep 2004. ISSN 1573-1405. doi: 10.1023/B:VISI.0000022288.19776.77. URL https://doi.org/10.1023/B:VISI.0000022288.19776.77.

Hugo Furtado, Marco Gonçalves, Nuno Fernandes, ; Paulo, and Sequeira Gonçalves. ESTUDO DOS POSTOS DE TRABALHO DE INSPECÇÃO DE DEFEITOS DA INDUSTRIA TÊXTIL. Technical report, 2001.

Rohith Gandhi. R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms, 2018. URL https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e. Último acesso em 18 de Fevereiro de 2020.

Ross Girshick. Fast r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

César Gonçalves. Como É que É testada a resistÊncia dos bancos dos carros?, 2019. URL https://www.motorspot.pt/2019/01/09/bancos-dos-carros-resistencia/. Último acesso em 20 de Novembro de 2019.

GrupoJBtecidos. Tecidos para automóveis – quais são e para quais veículos servem?, 2016. URL http://jbrevestimentos.com.br/blog/tecidos-para-automoveis-tapecaria-automotiva/. Último acesso em 19 de Novembro de 2019.

L Canan Dülger Mehmet Topalbekiroğlu H İbrahim Çelik. Fabric defect detection using linear filtering and morphological operations. *Indian Journal of Fibre Textile Research*, 39(5):469–483, 2014.

Kazım Hanbay, Muhammed Fatih Talu, and Ömer Faruk Özgüven. Fabric defect detection systems and methods—A systematic literature review. *Optik*, 127(24):11960–11973, dec 2016. ISSN 0030-4026. doi: 10.1016/J.IJLEO.2016.09.110. URL https://www.sciencedirect.com/science/article/pii/S0030402616311366.

Guang Hua Hu. Optimal ring Gabor filter design for texture defect detection using a simulated annealing algorithm. In *Proceedings - 2014 International Conference on Information Science, Electronics and Electrical Engineering, ISEEE 2014*, volume 2, pages 860–864. Institute of Electrical and Electronics Engineers Inc., nov 2014. ISBN 9781479931965. doi: 10.1109/InfoSEEE.2014.6947789.

Jonathan Hui. Ssd object detection: Single shot multibox detector for real-time processing, 2018. URL https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06. Último acesso em 1 de Junho de 2020.

Clemente Ibarra-Castanedo, José Ricardo Tarpani, and Xavier P V Maldague. Nondestructive testing with thermography. *European Journal of Physics*, 34(6):S91–S109, oct 2013. doi: 10.1088/0143-0807/34/6/s91. URL https://doi.org/10.1088%2F0143-0807%2F34%2F6%2Fs91.

Pandia Rajan Jeyaraj and Edward Rajan Samuel Nadar. Computer vision for automatic detection and classification of fabric defect employing deep learning algorithm. *International Journal of Clothing Science and Technology*, aug 2019. ISSN 17585953. doi: 10.1108/IJCST-11-2018-0135.

Junfeng Jing, Huanhuan Zhang, Jing Wang, Pengfei Li, and Jianyuan Jia. Fabric defect detection using Gabor filters and defect classification based on LBP and Tamura method. *Journal of the Textile Institute*, 104(1):

18–27, jan 2013. ISSN 0040-5000. doi: 10.1080/00405000.2012.692940. URL http://www.tandfonline.com/doi/abs/10.1080/00405000.2012.692940.

Raimi Karim. Illustrated: 10 cnn architectures, 2019. URL https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d. Último acesso em 23 de Janeiro de 2020.

Ayoosh Kathuria. What's new in yolo v3?, 2019. URL https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b. Último acesso em 26 de Maio de 2020.

Achraf KHAZRI. Faster rcnn object detection, 2019. URL https://towardsdatascience.com/faster-rcnn-object-detection-f865e5ed7fc4. Último acesso em 18 de Fevereiro de 2020.

Rico Krueger, Taha H. Rashidi, and John M. Rose. Preferences for shared autonomous vehicles. *Transportation Research Part C: Emerging Technologies*, 69:343–355, aug 2016. ISSN 0968090X. doi: 10.1016/j.trc.2016.06.015.

Harshall Lamba. Understanding semantic segmentation with unet, 2019. URL https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47. Último acesso em 1 de Junho de 2020.

Alfredo Lavrador. Nem imagina o que destrói o interior do seu carro, 2019. URL https://observador.pt/2019/06/16/nem-imagina-o-que-destroi-o-interior-do-seu-carro/. Último acesso em 20 de Novembro de 2019.

Rebecca Littman and Elizabeth Levy Paluck. The Cycle of Violence: Understanding Individual Participation in Collective Violence. *Political Psychology*, 36:79–99, feb 2015. ISSN 0162895X. doi: 10.1111/pops.12239. URL http://doi.wiley.com/10.1111/pops.12239.

Junyan Liu, Wang Yang, and Jingmin Dai. Research on thermal wave processing of lock-in thermography based on analyzing image sequences for NDT. *Infrared Physics and Technology*, 53(5):348–357, 2010. ISSN 13504495. doi: 10.1016/j.infrared.2010.06.002. URL http://dx.doi.org/10.1016/j.infrared.2010.06.002.

Li Liu, Jianhong Zhang, Xiaodong Fu, Lijun Liu, and Qingsong Huang. Unsupervised segmentation and elm for fabric defect image classification. *Multimedia Tools and Applications*, 78(9):12421–12449, may 2019. ISSN 15737721. doi: 10.1007/s11042-018-6786-7.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9905 LNCS:21–37, dec 2015. doi: 10.1007/978-3-319-46448-0_2. URL http://arxiv.org/abs/1512.02325http://dx.doi.org/10.1007/978-3-319-46448-0{_}2.

Abdel Salam Malek, Jean-Yves Drean, Laurent Bigue, and Jean-François Osselin. Optimization of automated online fabric inspection by fast Fourier transform (FFT) and cross-correlation. *Textile Research Journal*, 83(3):256–268, feb 2013. ISSN 0040-5175. doi: 10.1177/0040517512458340. URL http://journals.sagepub.com/doi/10.1177/0040517512458340.

Roberto Montanini. Quantitative determination of subsurface defects in a reference specimen made of Plexiglas by means of lock-in and pulse phase infrared thermography. *Infrared Physics and Technology*, 53(5): 363–371, 2010. ISSN 13504495. doi: 10.1016/j.infrared.2010.07.002. URL http://dx.doi.org/10.1016/j.infrared.2010.07.002.

Arthur Ouaknine. Review of deep learning algorithms for object detection, 2018. URL https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852. Último acesso em 18 de Fevereiro de 2020.

Prabhu Raghav. Understanding of convolutional neural network (cnn) — deep learning, 2018. URL https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. Último acesso em 22 de Janeiro de 2020.

Jagdish Lal Raheja, Sunil Kumar, and Ankit Chaudhary. Fabric defect detection based on GLCM and Gabor filter: A comparison. *Optik*, 124(23):6469–6474, 2013. ISSN 00304026. doi: 10.1016/j.ijleo.2013.05.004.

Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. apr 2018. URL http://arxiv.org/abs/1804.02767.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9351, pages 234–241. Springer Verlag, may 2015. ISBN 9783319245737. doi: 10.1007/978-3-319-24574-4_28.

Adrian Rosebrock. Intersection over union (iou) for object detection, 2016. URL https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. Último acesso em 12 de Fevereiro de 2020.

Abhinav Sagar. Deep learning for image classification with less data, 2019. URL https://towardsdatascience.com/deep-learning-for-image-classification-with-less-data-90e5df0a7b8e. Último acesso em 20 de Fevereiro de 2020.

Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way, 2018. URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Último acesso em 23 de Janeiro de 2020.

Gal Shenhav. Convolutional neural network architecture: Forging pathways to the future. URL https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-architecture-forging-pathways-future/. Último acesso em 12 de Junho de 2020.

Surya Pratap Singh. Fully connected layer: The brute force layer of a machine learning model, 2017. URL https://iq.opengenus.org/fully-connected-layer/. Último acesso em 23 de Janeiro de 2020.

Tamás Tettamanti, István Varga, and Zsolt Szalay. Impacts of autonomous cars from a traffic engineering perspective, oct 2016. ISSN 15873811.

Marvin Tortas. Como verificar os sinais de desgaste do carro em 14 passos, 2019. URL https://www.e-konomista.pt/como-verificar-os-sinais-de-desgaste-do-carro/. Último acesso em 20 de Novembro de 2019.

Sik-Ho Tsang. Review: Ssd — single shot detector (object detection), 2018. URL https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11. Último acesso em 2 de Junho de 2020.

Sik-Ho Tsang. Review: Deeplabv3+ — atrous separable convolution (semantic segmentation), 2019a. URL https://medium.com/@sh.tsang/review-deeplabv3-atrous-separable-convolution-semantic-segmentation-a625f6e83b90. Último acesso em 23 de Janeiro de 2020.

Sik-Ho Tsang. Review: Yolov3 — you only look once (object detection), 2019b. URL https://towardsdatascience.com/review-yolov3-you-only-look-once-object-detection-eab75d7a1ba6. Último acesso em 27 de Maio de 2020.

Muneeb ul Hassan. R-cnn – neural network for object detection and semantic segmentation, 2018a. URL https://neurohive.io/en/popular-networks/r-cnn/. Último acesso em 14 de Fevereiro de 2020.

Muneeb ul Hassan. Vgg16 – convolutional network for classification and detection, 2018b. URL https://neurohive.io/en/popular-networks/vgg16/. Último acesso em 12 de Junho de 2020.

Lilian Weng. Object detection for dummies part 1: Gradient vector, hog, and ss. *lilianweng.github.io/lil-log*, 2017. URL http://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html.

Daniel Yapi, Mohand Said Allili, and Nadia Baaziz. Automatic Fabric Defect Detection Using Learning-Based Local Textural Distributions in the Contourlet Domain. *IEEE Transactions on Automation Science and Engineering*, 15(3):1014–1026, jul 2018. ISSN 15455955. doi: 10.1109/TASE.2017.2696748.